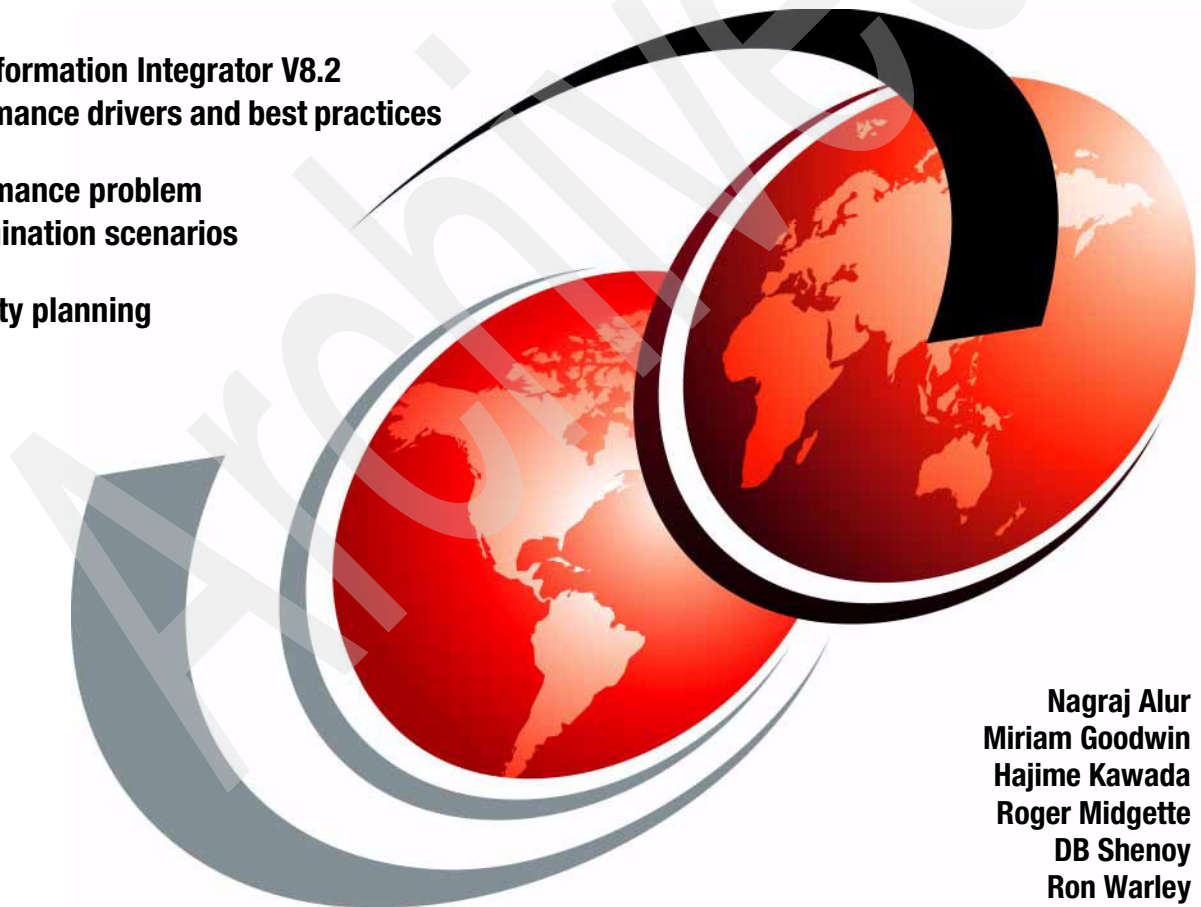


DB2 II: Performance Monitoring, Tuning and Capacity Planning Guide

DB2 Information Integrator V8.2
performance drivers and best practices

Performance problem
determination scenarios

Capacity planning



Nagraj Alur
Miriam Goodwin
Hajime Kawada
Roger Midgette
DB Shenoy
Ron Warley



International Technical Support Organization

DB2 II: Performance Monitoring, Tuning and Capacity Planning Guide

November 2004

Archived

Note: Before using this information and the product it supports, read the information in “Notices” on page xv.

First Edition (November 2004)

This edition applies to Version 8, Release 2 of DB2 Information Integrator (product number 5724-C74).

© Copyright International Business Machines Corporation 2004. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures	vii
Tables	ix
Examples	xi
Notices	xv
Trademarks	xvi
Preface	xvii
The team that wrote this redbook	xvii
Become a published author	xx
Comments welcome	xx
Chapter 1. DB2 Information Integrator architecture overview	1
1.1 Introduction	2
1.2 Current business trends	2
1.2.1 From on demand to grid computing	3
1.2.2 From grid computing to data federation	4
1.2.3 From data federation to information integration	5
1.3 IBM's DB2 Information Integration overview	6
1.3.1 Data consolidation or placement	8
1.3.2 Distributed access (federation)	9
1.3.3 DB2 Information Integrator products	9
1.4 DB2 Information Integrator V8.2	11
1.4.1 DB2 II V8.2 overview	11
1.4.2 DB2 II components	15
1.4.3 Configuring the federated system	18
1.4.4 Performance considerations	24
1.5 DB2 Information Integrator topology considerations	26
1.5.1 Dedicated federated server	28
1.5.2 Collocated federated server	28
Chapter 2. Introduction to performance management	31
2.1 Introduction	32
2.2 Performance management	32
2.3 Types of monitoring	35
2.3.1 Routine monitoring	35
2.3.2 Online/realtime event monitoring	36

2.3.3 Exception monitoring.	37
2.4 Problem determination methodology.	37
Chapter 3. Key performance drivers of DB2 II V8.2	41
3.1 Introduction	42
3.2 Compilation flow of a federated query.	44
3.3 Execution flow of a federated query	50
3.4 Key performance drivers	54
3.4.1 Performance factors	57
3.4.2 Federated server considerations.	60
3.4.3 Data source considerations.	102
3.4.4 Efficient SQL queries	109
3.4.5 Hardware and network	112
Chapter 4. Performance problem determination scenarios	115
4.1 Introduction	116
4.2 DB2 II hypotheses hierarchy	119
4.2.1 DB2 II federated database server resource constraints	123
4.2.2 DB2 II resource constraints.	124
4.2.3 Federated server or remote data source.	136
4.2.4 Federated server related.	152
4.2.5 Remote data source related	161
4.3 Monitoring best practices	162
4.3.1 Performance considerations	164
4.3.2 Best practices	165
4.4 Problem scenarios.	167
4.4.1 Federated test environment	167
4.4.2 Missing or incorrect statistics/index information	170
4.4.3 Poorly tuned sort heap and buffer pools	206
4.4.4 Missing or unavailable MQTs	210
4.4.5 Incompatible data types on join columns	239
4.4.6 Pushdown problems	272
4.4.7 Default DB2_FENCED wrapper option with DPF	339
Chapter 5. Capacity planning in an existing DB2 II environment	377
5.1 Introduction	378
5.2 Capacity planning assumptions.	378
5.3 Capacity planning procedure.	379
5.3.1 Capacity planning procedure overview	381
5.4 Capacity planning new applications	427
5.4.1 Model of different profiles of queries.	427
5.4.2 Determine new application workload	428
5.4.3 Estimate capacity for the new application	428

Appendix A. DB2 II V8.2 performance enhancements	429
Introduction	430
Fenced wrappers	430
Parallelism enhancements	432
Intra-partition parallelism in a non-DPF environment	432
Inter-partition parallelism in a DPF environment with local data	433
Inter-partition parallelism in a DPF environment without local data	434
Updating nickname statistics	436
Cache tables	436
Informational constraints	438
Snapshot monitor support	439
Health Center alerts	439
 Appendix B. DB2 EXPLAIN facility with DB2 Information Integrator	441
Brief review of the DB2 EXPLAIN facility	442
db2exfmt overview	448
EXPLAIN INSTANCE section	449
SQL STATEMENT section	450
Access plan graph	453
OPERATOR DETAILS section	455
Objects section	457
Complete db2exfmt output	458
Federated test environment	462
db2exfmt examples involving DB2 II	463
Join of nicknames referencing Oracle and SQL server	469
INTRA_PARALLEL = YES (intra-partition enabled)	489
Database Partition Feature (DPF) with FENCED = 'N'	516
Database Partition Feature (DPF) with FENCED = 'Y'	534
DB2_MAXIMAL_PUSHDOWN = 'N'	554
DB2_MAXIMAL_PUSHDOWN = 'Y'	586
SQL INSERT/UPDATE/DELETE	615
 Related publications	631
IBM Redbooks	631
Other publications	631
Online resources	632
How to get IBM Redbooks	633
Help from IBM	633
 Index	635

Archived

Figures

1-1	Data federation concept	5
1-2	Overview of IBM products for information integration	10
1-3	Data federation technology	11
1-4	Components of a federated system	13
1-5	Basic steps in configuring a federated system	19
1-6	DB2 II topologies	27
2-1	Performance management cycle	34
2-2	A typical problem determination methodology	38
3-1	Federated query processing flow	43
3-2	SQL Compiler query analysis flow	44
3-3	Query execution flow	52
3-4	Performance considerations overview	58
3-5	Federated query performance elements	59
3-6	Federated server considerations topics	61
3-7	Wrapper architecture - Fenced and trusted	63
3-8	Query optimization topics	68
3-9	Nickname statistics collected by data source	72
3-10	MQT/AST look-aside concept	86
3-11	Cache table concept	88
3-12	Overflowed sorts	93
3-13	Non-overflowed piped sorts	94
3-14	Database manager snapshot showing sort monitor elements	97
3-15	Database snapshot showing sort monitor elements	97
4-1	A typical DB2 II environment and hypotheses hierarchy	117
4-2	DB2 II hypotheses hierarchy	120
4-3	Triggering event determines entry into DB2 II hypotheses hierarchy	122
4-4	Statistics Update in DB2 Control Center	153
4-5	DB2 snapshot monitor syntax and data collection	163
4-6	Federated test environment	168
4-7	TPCD tables	169
4-8	ping DB2 II federated server and Oracle data source server (azov)	172
4-9	vmstat command output from the federated server	173
4-10	iostat command output from federated server	174
4-11	lsps command output from the federated server	174
4-12	ps -ef command output from the federated server	175
4-13	ps aux command from federated server	175
5-1	Capacity planning procedure overview	382
5-2	Contents of FEDWH.FEDWH_SNAPSHOT_DYN_SQL table (1 of 3)	393

5-3	Contents of FEDWH.FEDWH_SNAPSHOT_DYN_SQL table (2 of 3).	394
5-4	Contents of FEDWH.FEDWH_SNAPSHOT_DYN_SQL table (3 of 3).	395
5-5	FEDWH.FEDWH_SNAPSHOT_DYN_INTERVAL table (1 of 4)	407
5-6	FEDWH.FEDWH_SNAPSHOT_DYN_INTERVAL table (2 of 4).	408
5-7	FEDWH.FEDWH_SNAPSHOT_DYN_INTERVAL table (3 of 4)	409
5-8	FEDWH.FEDWH_SNAPSHOT_DYN_INTERVAL table (4 of 4)	410
5-9	Contents of FEDWH.FEDWH_FEDSQL_INTERVAL (1 of 2)	413
5-10	Contents of FEDWH.FEDWH_FEDSQL_INTERVAL (2 of 2)	414
5-11	Contents of FEDWH.FEDWH_INSTANCE_REPORT	417
5-12	Contents of FEDWH.FEDWH_FEDSQL_REPORT table (1 of 2)	418
5-13	Contents of FEDWH.FEDWH_FEDSQL_REPORT table (2 of 2)	419
5-14	Chart of maximum connections per monitoring interval.	423
5-15	Chart of number of executions per query.	424
A-1	Wrapper architecture - Fenced and trusted	431
A-2	Intra-partition parallelism on SMP systems	432
A-3	Inter-partition parallelism in a DPF environment with local data	433
A-4	Inter-partition parallelism in DPF environment with nickname data . . .	435
A-5	Cache table concept	437
B-1	Relationship of the main EXPLAIN tables	446

Tables

1-1	Key global catalog contents for remote data sources	16
3-1	Key performance drivers - non-DPF DB2 in relation to DB2 II.	55
4-1	Typical problem areas associated with DB2 II performance	118
4-2	Join strategies	158
4-3	Snapshot monitor switches	162
4-4	TPCD tables cardinality	169
5-1	Memory utilization versus maximum concurrent connections	426
5-2	Query profile model	428
B-1	DB2 EXPLAIN facility.	443
B-2	EXPLAIN tables	444
B-3	Operators in the access plan	446
B-4	db2exfmt output focus areas	463
B-5	DB2 II server options	464

Archived

Examples

3-1	User query involving join of two nicknames	65
3-2	db2exfmt output showing no CPG exploitation	65
3-3	db2exfmt output showing CPG exploitation	66
3-4	NNSTAT stored procedure	69
3-5	Database snapshot showing intra_parallelism monitor element	76
3-6	Snapshot showing block remote cursor information	101
3-7	db2exfmt output with query fragment in RMTQTEXT of SHIP operator.	102
3-8	Writing outer joins more efficiently	111
4-1	dbm snapshot for connections	126
4-2	db snapshot for connections	127
4-3	dbm snapshot for sorting	128
4-4	db snapshot for sorting	130
4-5	db snapshot for locking	131
4-6	db snapshot for buffer pools	132
4-7	db snapshot for catalogcache_sz and pckcachesz	134
4-8	Dynamic SQL snapshot	136
4-9	Sample db2exfmt output with a SHIP operator and RMTQTEXT field.	138
4-10	DBM CFG parameter settings affecting connections.	176
4-11	DB CFG parameter settings affecting connections	177
4-12	DBM snapshot for connection information	177
4-13	DB snapshot for connection information	178
4-14	DB and DBM CFG parameters affecting sorting	179
4-15	DBM snapshot for sorting information	179
4-16	DB snapshot for sorting information.	179
4-17	Buffer pool snapshot information	180
4-18	Application snapshot	182
4-19	Problem query	184
4-20	Dynamic SQL snapshot	184
4-21	db2exfmt output for the problem query	186
4-22	db2batch command	194
4-23	db2batch output	194
4-24	Update nickname statistics from command line.	197
4-25	Nickname index specifications.	197
4-26	db2exfmt after nickname statistics updated.	198
4-27	Routine monitoring snapshot information	206
4-28	DB and DBM CFG parameters affecting sorting	208
4-29	Default BP size	208
4-30	DBM snapshot for sorting information	208

4-31	Adjust SHEAPTHRES and SORTHEAP configuration parameters . . .	209
4-32	Problem query	211
4-33	Dynamic SQL snapshot	212
4-34	db2exfmt output for the problem query	214
4-35	MQT definition for ORDERSMQT	229
4-36	db2exfmt output with MQT	230
4-37	Dynamic SQL snapshot with ORDERSMQT	238
4-38	Problem query	240
4-39	Dynamic SQL snapshot	240
4-40	db2exfmt output for the problem query	242
4-41	Data types for nickname join columns before ALTER	259
4-42	Statistics for nickname join columns before ALTER	261
4-43	ALTER NICKNAME statement.	262
4-44	Data types for nickname join columns after ALTER	262
4-45	Statistics for nickname join columns after ALTER	262
4-46	Manually update HIGH2KEY and LOW2KEY values.	263
4-47	db2exfmt of problem query after fixing mismatched data types.	264
4-48	Problem query	273
4-49	Dynamic SQL snapshot	274
4-50	db2exfmt output for DB2_MAXIMAL_PUSHDOWN = 'N'	278
4-51	Alter the wrapper option DB2_MAXIMAL_PUSHDOWN to 'Y'	310
4-52	db2exfmt output for DB2_MAXIMAL_PUSHDOWN = 'Y'	310
4-53	Problem query	340
4-54	Dynamic SQL snapshot	340
4-55	db2exfmt output of problem query	341
4-56	Show wrapper option	355
4-57	Alter the wrapper option DB2_FENCED to 'Y'	355
4-58	db2exfmt output of problem query	356
5-1	Set the DB2 monitor switches	382
5-2	Create the EXPLAIN tables	383
5-3	Create performance warehouse tables	384
5-4	Bind db2batch with CS isolation level	386
5-5	DBM CFG parameter settings affecting connections.	386
5-6	Checking the state of the package cache	387
5-7	Capture dynamic SQL snapshot into performance warehouse table.	387
5-8	Dynamic SQL snapshot	388
5-9	Output of the operating system sar command	391
5-10	Output of the operating system vmstat command	391
5-11	High water mark value of memory utilization	392
5-12	Capture dynamic SQL snapshot into performance warehouse table.	392
5-13	Snapshot for maximum concurrent connections	395
5-14	Insert sar and vmstat info into FEDWH.FEDWH_INSTANCE table.	396
5-15	Populate the EXPLAIN tables	397

5-16	Sample db2exfmt output with a SHIP operator and RMTQTXT field . . .	398
5-17	Link EXPLAIN output to particular monitoring interval using SNAPID .	405
5-18	Compute metrics for the monitoring interval and store	405
5-19	Summarize monitored intervals	411
5-20	Populating the utilization reports	416
5-21	db2 batch output	420
B-1	Sample federated SQL statement	448
B-2	EXPLAIN INSTANCE section	449
B-3	STATEMENT section	450
B-4	Access Plan section	453
B-5	OPERATOR DETAILS section	455
B-6	Objects section	457
B-7	db2exfmt output for the sample federated statement	458
B-8	db2exfmt output - Join of nicknames	469
B-9	db2exfmt output for intra-partition parallelism enabled	490
B-10	db2exfmt output for trusted wrapper	516
B-11	db2exfmt output for fenced wrapper	534
B-12	db2exfmt output for DB2_MAXIMAL_PUSHDOWN = 'N'	554
B-13	db2exfmt output for DB2_MAXIMAL_PUSHDOWN = 'Y'	586
B-14	db2exfmt output for SQL INSERT	615
B-15	db2exfmt output for SQL UPDATE	624
B-16	db2exfmt output for SQL DELETE	627

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®
DB2 Connect™
DB2 Universal Database™
DB2®
DRDA®
ibm.com®
Informix®

IBM®
IMS™
iSeries™
OS/390®
pSeries®
Redbooks™
Redbooks (logo) ™

Tivoli®
WebSphere®
xSeries®
z/OS®
zSeries®

The following terms are trademarks of other companies:

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Preface

This IBM Redbook will help you develop, monitor, tune, and size DB2® Information Integrator Version 8.2 applications in the AIX® and Windows® environments.

This publication is aimed at DBAs responsible for managing the performance of a DB2 Information Integrator environment.

This publication is organized as follows:

- ▶ **Chapter 1** provides an overview of DB2 Information Integrator, and describes the pros and cons of implementing different configurations in a typical customer environment.
- ▶ **Chapter 2** provides an overview of performance management in general, as well as specific considerations that apply to a DB2 Information Integrator environment.
- ▶ **Chapter 3** discusses the key performance drivers of a DB2 Information Integrator environment, and provides best practices recommendations for achieving optimal performance. This will include a description of the tools available to monitor and tune DB2 Information Integrator environments.
- ▶ **Chapter 4** describes a rigorous approach to performance problem diagnosis in a DB2 Information Integrator environment, and applies this approach in diagnosing a number of commonly encountered performance problems in typical customer environments.
- ▶ **Chapter 5** describes an approach for performing capacity planning of an existing DB2 Information Integrator environment.
- ▶ **Appendix A** provides an overview of DB2 Information Integrator Version 8.2 performance enhancements.
- ▶ **Appendix B** provides EXPLAIN output of various queries in a federated environment, and discusses key items in the EXPLAIN output to focus on from a performance perspective.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, San Jose Center.

Nagraj Alur is a Project Leader with the IBM® International Technical Support Organization, San Jose Center. He holds a Masters Degree in Computer Science from the Indian Institute of Technology (IIT), Mumbai, India. He has more than 28 years of experience in DBMSs, and has been a programmer, systems analyst, project leader, consultant, and researcher. His areas of expertise include DBMSs, data warehousing, distributed systems management, and database performance, as well as client/server and Internet computing. He has written extensively on these subjects and has taught classes and presented at conferences all around the world. Before joining the ITSO in November 2001, he was on a two-year assignment from the Software Group to the IBM Almaden Research Center, where he worked on Data Links solutions and an eSourcing prototype.

Miriam Goodwin is an IT Specialist in Dallas, TX specializing in technical sales support for data management products. She has worked with DB2 UDB on all platforms for several years in technical sales support, implementation services, database administration, and application programming roles. Miriam has worked at IBM for nine years, and with DB2 Information Integrator since the original release of the Data Joiner product in the mid-1990s. Prior to joining IBM, Miriam worked with DB2 and other IBM database management products for 10 years for various IBM customers in a variety of roles.

Hajime Kawada is an IT Specialist providing technical sales support for information management products in Japan, with particular emphasis on DB2 Information Integrator. He has conducted performance and functional tests with DB2 Information Integrator, and has taught seminars on it. Prior to joining IBM, he had five years of experience developing solutions using Oracle and other databases.

Roger Midgette is a Sr. Systems Engineer with Maricom Systems, Incorporated, a Maryland-based Information Technology solutions provider. Roger has over 25 years experience with large-scale enterprise communications and database systems. His successful long-term career with IBM and subsequent positions with Mitre Corporation and The Fillmore Group have provided him with experience in a variety of technical marketing, product development, technical support, and training roles. Roger's broad background has allowed him to assist national, international, governmental, and commercial organizations in implementing complex data management solutions. Roger is currently focusing on database middleware, including DB2 Information Integrator and DB2 Connect™, utilizing his understanding of the entire IT environment. He has presented data management seminars and classes, published DB2 technical documents, and authored DB2 Internet-based tutorials. He holds a Masters Degree in Information Technology from the University of Maryland and is an IBM DB2 UDB Certified Solutions Expert - Database Administration and DRDA®.

DB Shenoy is a Certified Consulting Software IT Specialist for IBM Information Management software, based in Menlo Park, CA. He has over 12 years of experience in relational database systems and holds a Bachelor's degree in Computer Science from Karnatak University, India. DB has extensive experience with DB2 UDB, Information Integrator, and Informix® technologies.

Ron Warley has spent the last 23 years working in IBM technical software support, both as a Systems Engineer for 10 years and in his current role as a Certified Consulting Software IT Specialist for 13 years. Prior to IBM, Ron worked for the US Government (Army) for eight years in Information Technology doing complex data simulation for the Department of Defense. Ron has had extensive DB2 experience as an IBM Systems Engineer and Software IT Specialist on many platforms: Linux and Windows NT/2000 on iSeries™, zSeries®, SUN Solaris, pSeries® AIX, and HP-UX. Ron has several DB2 product certifications, and specializes in DB2 Business Intelligence solutions with DB2 middleware products such as DB2 Information Integrator.

Thanks to the following people for their significant contributions to this project:

Anjali Betawadkar-Norwood
Aakash Bordia
Susanne Englert
Farnaz Erfan
Doreen Fogle
Lan Huang
Eileen Lin
Robin Noble-Thomas
Kiran Potu
Micks Purnell
Yang Sun
Ioana-mihaela Ursu
Swati Vora
Tian Zhang
IBM Silicon Valley Laboratory

Simon Harris
IBM United Kingdom

Xiaoyan Qian
Calisto Zuzarte
IBM Toronto Laboratory

Bertrand Dufrasne
Bart Steegmans
International Technical Support Organization, San Jose Center

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

ibm.com/redbooks

- ▶ Send your comments in an Internet note to:

redbook@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. QXXE Building 80-E2
650 Harry Road
San Jose, California 95120-6099



DB2 Information Integrator architecture overview

In this chapter we briefly describe the business needs driving the need for information integration, and IBM's response to this demand with its DB2 Information Integrator family of products. We introduce the IBM DB2 Information Integrator family of products and focus on DB2 Information Integrator since it is the focus of this publication.

The topics covered are:

- ▶ Current business trends
- ▶ IBM's DB2 Information Integration overview
- ▶ DB2 Information Integrator V8.2
- ▶ DB2 Information Integrator topology considerations

1.1 Introduction

A number of business trends are driving the need for integration of data and processes across employees, customers, business partners, and suppliers. The inherent heterogeneity of hardware and software platforms in intranets and extranets presents unique challenges that must be overcome in order to gain a competitive advantage in the global economy.

In this chapter we discuss the current business trends fueling integration demands, IBM's DB2 Information Integrator solution, and IBM's federated DB2 Information Integrator V8.2 offering.

1.2 Current business trends

To keep up with the evolution of e-business computing, companies in every industry are being challenged to act—and react—on demand. Responding to any customer demand, each market opportunity and every external threat requires integration between people, processes, and information—this integration must extend across the company, and across partners, suppliers, and customers.

Integration, automation, and virtualization are the three key elements of this on-demand operating environment:

- ▶ **Integration** is the efficient and flexible combination of data to optimize operations across and beyond the enterprise. It is about people, processes, and information.
- ▶ **Automation** is the capability to increasingly automate business processes with the ultimate goal of self-regulation, thereby reducing the complexity of data management to enable better use of assets.
- ▶ **Virtualization** provides a single, consolidated view of and easy access to all available resources in a network, no matter where the data resides, or the type of data source.

Note: IBM defines an on demand business as an enterprise whose business processes integrate end-to-end across the company with key partners, suppliers, and customers in order to respond with speed to any customer demand, market opportunity, or external threat.

IBM has identified five types of integration that are based on an open services infrastructure. You can use these types of integration together or separately to solve business issues. The following five types of integration represent the

various integration challenges that face businesses today—with information integration being at the core of these integration types.

- ▶ User interaction

A user can work with a single tailored user interface, which is available through virtually any device, with full transactional support. The results of the user's interaction are integrated into multiple business systems.

- ▶ Process integration

A business can change how it operates through modeling, automation, and monitoring of processes across people and heterogeneous systems—both inside and outside the enterprise.

- ▶ Application connectivity

Applications can connect to one another so that they share and use information for better use at the enterprise level.

- ▶ Build to integrate

Users can build and deploy integration-ready applications by using Web services and existing assets. You can integrate new solutions with existing business assets.

- ▶ Information integration

Diverse forms of business information can be integrated across the enterprise. Integration enables coherent search, access, replication, transformation, and analysis over a unified view of information assets to meet business needs.

In the following subsections, we describe how the success of an on demand business enterprise is significantly dependent upon a seamless and scalable information integration infrastructure.

- ▶ From on demand to grid computing
- ▶ From grid computing to data federation
- ▶ From data federation to information integration

1.2.1 From on demand to grid computing

Grid computing is distributed computing taken to the next evolutionary level. The grid provides an infrastructure on which to support a large collection of communication resources, such as hardware and software.

The standardization of communications between heterogeneous systems created the Internet explosion. The emerging standardization for sharing resources, along with the availability of higher bandwidth, is driving a potentially equally large evolutionary step in grid computing.

One major function of the grid is to better balance resource utilization.

An organization may have occasional unexpected peaks of activity that demand more resources. If the applications are grid enabled, the application workload can be moved to under-utilized machines during such peaks. In general, a grid can provide a consistent way to balance workloads on a wider federation of resources.

1.2.2 From grid computing to data federation

An increasing number of grid applications manage very large volumes of geographically distributed data. The complexity of data management on a grid is due to the scale, dynamism, autonomy, and distribution of data sources.

One way of accessing diverse business information from a variety of sources and platforms is through data federation.

Data federation is the ability to transparently access diverse business data from a variety of sources and platforms as though it were from a single resource. A federated server may access data directly, such as accessing a relational database, or accessing an application that creates and returns data dynamically (such as a Web service). Figure 1-1 on page 5 shows the federated approach to information integration as providing the ability to synchronize distributed data without requiring that it be moved to a central repository.

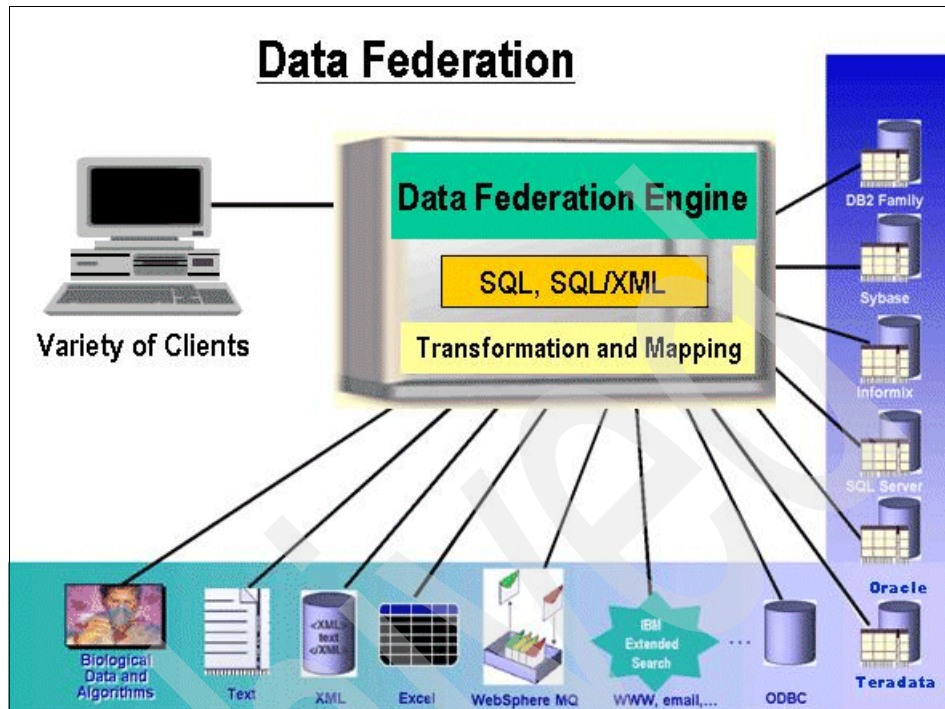


Figure 1-1 Data federation concept

Based on ongoing research investments and proven data management technologies in areas such as relational data, XML, content management, federation, search, and replication, IBM has developed the integrated infrastructure shown in Figure 1-1. Data federation uses SQL as the single language to access all data sources. This enables all data sources to be accessed in a standardized format, whether they are an application program, tool, or program product.

1.2.3 From data federation to information integration

Information integration builds on the solid foundation of existing data management solutions. Information integration provides an end-to-end solution for transparently managing both the volume and diversity of data that exists in enterprises and organizations today.

Increasingly business IT operations involve the need to integrate diverse and unconnected infrastructures.

The following goals are critical to increasing operations efficiency and gaining a competitive advantage:

- ▶ Integrate seamlessly with new businesses and link packaged applications with legacy systems.
- ▶ Control the accelerating costs of managing disparate systems and integrating across heterogeneous pockets of automation.
- ▶ Mitigate shortages of people and skills while quickly reaching new markets.
- ▶ Implement solutions that efficiently access and manage information across product and industry boundaries.

1.3 IBM's DB2 Information Integration overview

Today, any but the simplest of business tasks require the use of information from the variety of data sources that businesses have built over many years. These sources may be local or remote, on the intranet, extranet, or internet. The data may be stored in any of a variety of formats such as relational or non-relational databases, flat files, and unstructured content stores. The data may be current or point-in-time copies. Often, the users need both read and write access to these sources.

This complex and dynamic environment presents significant challenges to business users and applications, as well as to the IT people who must maintain and manage it.

Important: IBM's vision of information integration is to significantly reduce or even eliminate these issues.

The underlying principle of information integration is for users to be able to see all of the data they use as if it resided at a single source. Information integration technology shields the requester from all the complexities associated with accessing data in diverse locations, including connectivity, semantics, formats, and access methods. Using a standards-based language such as structured query language (SQL), extensible markup language (XML) through SQL/XML, or a standard Web services or content API, information integration middleware enables users, or applications acting on their behalf, to access information transparently without concern for its physical implementation.

The goal of providing an integrated view of information can be achieved in two ways, as follows:

1. **Data consolidation or placement**, which involves moving the data to a more efficient or accessible location

Consolidating data into a single physical store has been the best way to achieve fast, highly available, and integrated access to related information. Creating a single physical copy lets businesses meet access performance or availability requirements, deliver snapshots that are point-in-time consistent, and provide sophisticated transformation for semantic consistency. Consolidated data stores, which are typically managed to extract, transform, load (ETL) or replicate processes, are the standard choice for information integration today.

However, these consolidated stores have some drawbacks, as follows:

- They are expensive—racking up significant additional administration, server and storage costs.
- The latency between the copy and the source of record can be a problem when you need the most current data.
- Rich content such as documents, images, or audio is typically not included.

2. **Distributed access**, which involves providing distributed access to data through data access or federation

Distributed access corresponds to the emerging category of technology called enterprise information integration (EII), which addresses some of the shortfalls of data consolidation or placement. EII represents middleware technology that lets applications access diverse and distributed data as if it were a single source, regardless of location, format, or access language. Access performance will typically be slower than for consolidated stores because the query may have to gather information from distributed locations across the network rather than access a single, local copy of data.

However, the benefits of EII include:

- Reduced implementation and maintenance costs because you do not have the additional hardware (server and storage), skills, and personnel costs.
- Access to current data from the source of record.
- Combining traditional data with mixed-format data.
- Access to copy-prohibited data based on data security, licensing restrictions, or industry regulations that restrict data movement; for example, some European countries prohibit commingling a customer's personal data with account data in a single database. But you can materialize a single image of the data by federating them at the time of access.

Note: Distributed sources must be sufficiently consistent to make joining the data both possible and meaningful. There must be a key on which the data can be joined or correlated such as a customer identifier, and the joined data must represent related topics.

Both data consolidation or placement and distributed access data consolidation serve distinct problem domains and are very complementary. They may be used alone or together to form the heart of what is required to integrate information.

Both approaches require extensive and largely common supporting functionality. Neither distributed access nor data consolidated or placement can exist without mapping and transformation functionality, which ensure data integrity. Furthermore, depending on the business requirement, the same data may need to be consolidated in some cases and federated in others. Therefore, a common set of transformation and mapping functionality is required in both cases to maintain consistency across the data used by the business.

In the following sections, we briefly describe scenarios where data consolidation and distributed access are appropriate, and then provide an overview of DB2 Information Integration products.

1.3.1 Data consolidation or placement

Data consolidation or placement brings together data from a variety of locations into one place, in advance, so that a user query does not always need to be distributed. This approach corresponds to ETL and replication functionality. You can use ETL to build a warehouse, replication to keep it automatically updated on a scheduled basis, and extend it with federation for queries that require data that did not make sense to put in the warehouse.

Scenarios where ETL or replication approaches are appropriate include the following:

- ▶ Access performance or availability requirements demand centralized or local data.
- ▶ Complex transformation is required to achieve semantically consistent data.
- ▶ Complex, multidimensional queries are involved.
- ▶ Currency requirements demand point-in-time consistency, such as at the close of business.

1.3.2 Distributed access (federation)

Very simply, federation takes a query in one location and distributes the appropriate parts of it to act upon the data wherever and in whatever form it resides.

Scenarios where distributed access approaches are appropriate include the following:

- ▶ Access performance and load on source systems can be traded for an overall lower implementation cost.
- ▶ Data currency requirements demand a fresh copy of the data.
- ▶ Widely heterogeneous data.
- ▶ Rapidly changing data.
- ▶ Data security.
- ▶ Licensing restrictions, or industry regulations restrict data movement.
- ▶ Unique functions must be accessed at the data source.
- ▶ Queries returning small result sets among federated systems.
- ▶ Large volume data that are accessed infrequently.

1.3.3 DB2 Information Integrator products

IBM's Information Integrator solution consists of a number of products and technologies that fall under a solution umbrella called IBM DB2 Information Integrator, as shown in Figure 1-2 on page 10.

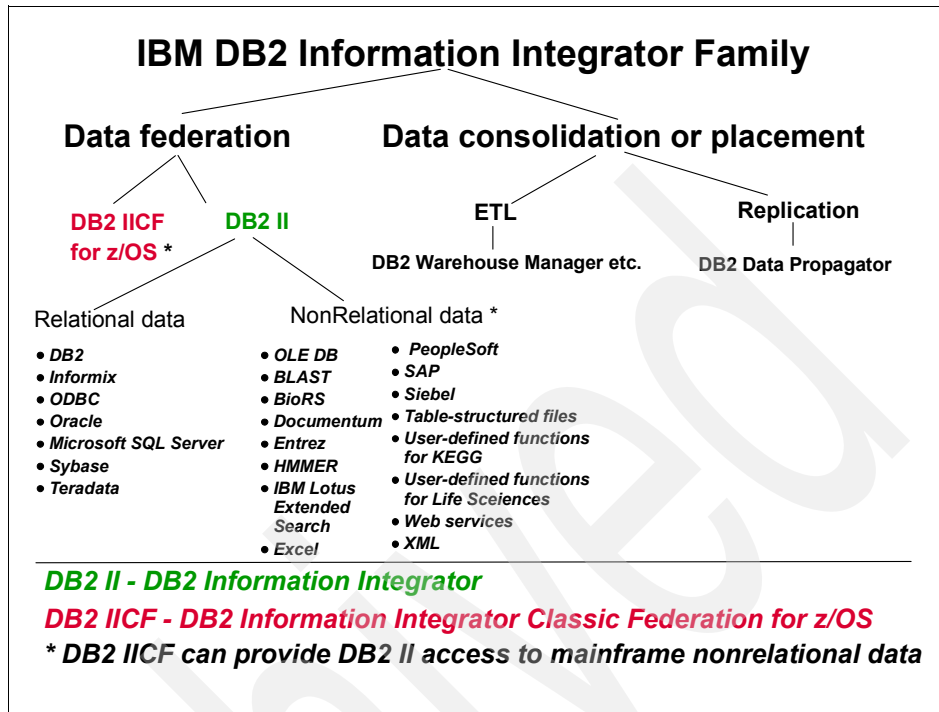


Figure 1-2 Overview of IBM products for information integration

There two main products that fall under the federation approach are:

► **DB2 Information Integrator (DB2 II)**

DB2 II is targeted at the application development community familiar with relational database application development. Applications that use SQL, or tools that generate SQL such as integrated development environments, and reporting and analytical tools, can now access and manipulate distributed and diverse data through a federated data server.

► **DB2 Information Integrator Classic Federation for z/OS® (DB2 IICF)**

DB2 IICF supports read/write access to relational and non-relational mainframe data sources such as IMS™, VSAM, Adabas, CA-IDMS, and CA-Datcom.

Note: This publication only focuses on DB2 II, hence the added detail in Figure 1-2.

1.4 DB2 Information Integrator V8.2

In this section we provide an overview of DB2 Information Integrator V8.2, describe its main components, discuss the main steps involved in configuring a data source, and provide a brief overview of some of the performance considerations.

The topics covered are:

- ▶ DB2 II V8.2 overview
- ▶ DB2 II components
- ▶ Configuring the federated system
- ▶ Performance considerations

1.4.1 DB2 II V8.2 overview

DB2 II's federated technology enables customers to abstract a common data model across diverse and distributed data and content sources, and to access and manipulate them as though they were a single source.

As mentioned earlier, with the data federation capability, the federated system acts as a virtual database with remote objects configured similar to local tables, as shown in Figure 1-3.

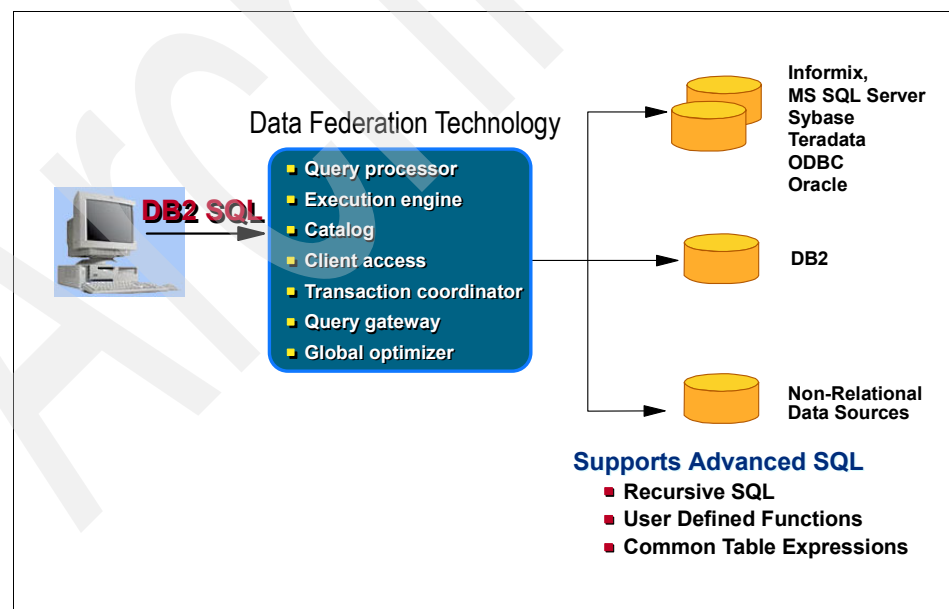


Figure 1-3 Data federation technology

With a federated system, you can send distributed requests to multiple data sources within a single SQL statement; for example, you can join data that is located in a DB2 UDB table, an Informix table, and an XML tagged file in a single SQL statement.

When an application submits a query to the federated system, the federated DB2 identifies the relevant data sources, and develops a query execution plan for obtaining the requested data. The plan typically breaks the original query into fragments that represent work to be delegated to individual data sources, as well as additional processing to be performed by the federated DB2 server to further filter, aggregate, or merge the data. The ability of the federated DB2 server to further process data received from sources allows applications to take advantage of the full power of the query language, even if some of the information requested comes from data sources with little or no native query processing capability, such as simple text files. The federated DB2 server has a local data store to cache query results for further processing if necessary.

A DB2 federated system is a special type of DBMS. A federated system consists of the following:

- ▶ A DB2 instance that operates as a federated server.
- ▶ A database that acts as the federated database for various relational and non-relational data sources.
- ▶ Clients (users and applications) that access the database and data sources. A nickname is the mechanism used by the clients to reference a remote data source object as if it were a local table.

The federated server communicates with the data sources by means of software modules called *wrappers*, as shown in Figure 1-4 on page 13.

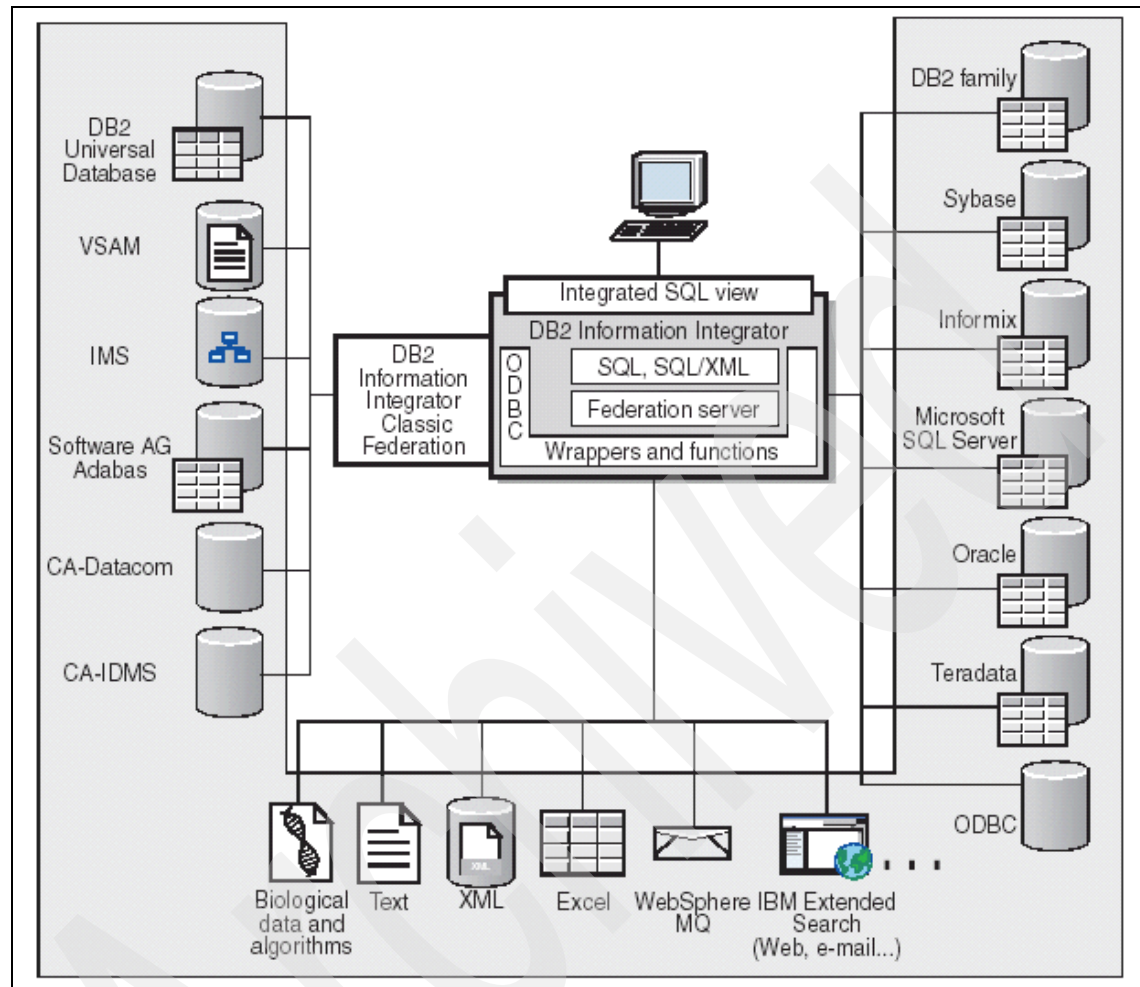


Figure 1-4 Components of a federated system

Wrappers are mechanisms by which the federated server interacts with the data sources. The federated server uses routines stored in a library called a *wrapper module* to implement a wrapper. These routines allow the federated server to perform operations such as connecting to a data source and retrieving data from it. The wrapper encapsulates data source information and models data as tables. It is aware of the characteristics of the data source, and it can expose unique

functions. A wrapper provides the programming logic to facilitate the following tasks:

1. Federated object registration

A wrapper encapsulates the data source characteristics from the federated engine. A wrapper knows what information is needed to register each type of data source.

2. Communication with the data source

Communication includes establishing and terminating connections with the data source, and maintaining the connection across statements within an application if possible.

3. Services and operations

Depending on the capabilities of the type of data sources that a wrapper is meant to access, different operations are supported. The operations can include sending a query to retrieve results, updating remote data, transaction support, large object manipulation, input value binding, compensation¹, and more.

4. Data modelling

A wrapper is responsible for mapping the data representation of the result of remote queries into the table format as required by the federated engine.

Wrappers are available for each type of data source. For example, if you want to access three DB2 for z/OS database tables, one DB2 for iSeries table, two DB2 UDB for Windows tables, two Informix tables, and one Informix view, you need to define only two wrappers: One for the DB2 data source objects and one for the Informix data source objects. Once these wrappers are registered in the federated database, you can use these wrappers to access other objects from those data sources.

DB2 Information Integrator V8.2 includes the ability to federate, search, cache, transform, and replicate data. As a federated data server, it provides out-of-the box access to DB2 Universal Database™; IBM Informix products; as well as databases from Microsoft®, Oracle, Sybase, and Teradata. In addition, it can also access semi-structured data from WebSphere® MQ messages, XML documents, Web services, Microsoft Excel, flat files, ODBC or OLE DB sources, plus a variety of formats unique to the life sciences industry. Integrated support for IBM Lotus® Extended Search provides the solution's broad content access to a variety of content repositories, including DB2 Content Manager, as well as

¹ Compensation is the ability by DB2 to process SQL that is not supported by a data source. DB2 compensates for lack of functionality at the data source in two ways: One way is to ask the data source to use one or more operations that are equivalent to the DB2 function stated in the query, and another way is to return the set of data from the data source to the federated server and perform the function locally.

e-mail databases, document repositories, third-party Internet search engines, and LDAP directories. DB2 Information Integrator Classic Federation for z/OS supports read/write access to relational and non-relational mainframe data sources such as IMS, VSAM, Adabas, CA-IDMS, and CA-Datacom.

Note: Applications can insert, update, or delete rows in federated relational databases; however, this is limited to single-site updates with only one-phase commits.

The power of DB2 II is in its ability to:

- ▶ Join data from local tables and remote data sources, as if all the data is stored locally in the federated database.
- ▶ Update data in relational data sources, as if the data is stored in the federated database.
- ▶ Replicate data to and from relational data sources.
- ▶ Take advantage of the data source processing strengths, by sending distributed requests to the data sources for processing.
- ▶ Compensate for SQL limitations at the data source by processing parts of a distributed request at the federated server.

Note: DB2 II V8.2 is supported on the Linux, UNIX®, and Windows platforms.

For an up-to-date and complete list of data sources supported, their corresponding data types, their corresponding versions, and the access method used by IBM DB2 Information Integrator V8.2 to access them, refer to the *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*, GC18-7036; and *IBM DB2 Information Integrator Federated Systems Guide Version 8.2*, SC18-7364.

The *IBM DB2 Information Integrator Data Source Configuration Guide* provides details on configuring access to each of the types of data sources supported by DB2 II.

ftp://ftp.software.ibm.com/ps/products/db2/info/vr82/pdf/en_US/iylse81.pdf

1.4.2 DB2 II components

DB2 II contains the following components:

- ▶ **DB2 UDB Enterprise Server Edition (ESE)** for Linux, UNIX, and Windows, which you can use to create and manage non-partitioned or partitioned database environments.

- The **relational wrappers** are used for non-IBM relational databases. In DB2 UDB Enterprise Server Edition (ESE) V8 for Linux, UNIX, and Windows, relational wrappers are required if you want to access data that is stored in Oracle, Sybase, Microsoft SQL Server, ODBC, and Teradata data sources.

Note: DB2 UDB ESE provides DB2, Informix, and OLE DB data sources federation without DB2 Information Integrator by setting the database manager configuration parameter FEDERATED to YES. DB2 II is only needed for federated access to the other relational sources identified.

- **Non-relational wrappers** are used by the DB2 federated system to integrate non-relational data sources, such as flat files and XML files; and genetic, chemical, biological, and other research data from distributed sources.
- **Global catalog** is the catalog in the federated database that holds information about the entire federated system. The global catalog holds information about the objects (tables, indexes, functions, etc.) in the federated database as well as information about objects (wrappers, remote servers, nicknames and their relationships) at the data sources. The information stored is about local and remote column names, column data types, column default values, and index information.

DB2 Information Integrator V8.2 *extends* the data federation technology already available in DB2 UDB for Linux, UNIX, and Windows.

The global catalog contains statistical information for nicknames, information on remote indexes for nicknames, and information on some attributes of each remote source, as shown in Table 1-1. It also it contains data type and function mappings.

Table 1-1 Key global catalog contents for remote data sources

Federated objects	Catalog views	Descriptions
Wrappers	SYSCAT.WRAPPERS SYSCAT.WRAPOPTIONS	Registered wrappers and their specific options (wraptype='R'/'N' for Relational/Non-relational wrapper).
Servers	SYSCAT.SERVERS SYSCAT.SERVEROPTIONS	Registered remote data sources and their specific options.
User mappings	SYSCAT.USEROPTIONS	Registered user authentications for specific servers for a DB2 user. The password setting is stored encrypted.

Federated objects	Catalog views	Descriptions
Nicknames	SYSCAT.TABLES SYSCAT.TABOPTIONS SYSCAT.COLUMNS, SYSCAT.COLOPTIONS SYSCAT.INDEXES SYSCAT.INDEXOPTIONS SYSCAT.KEYCOLUSE	Registered nicknames are identified with TYPE='N' in SYSCAT.TABLES. SYSCAT.TABOPTIONS stores specific options about nicknames. SYSCAT.COLOPTIONS stores specific options about nicknames; for instance, the server name, remote schema, and remote table name. SYSCAT.KEYCOLUSE stores information about primary key.
Index specifications	SYSCAT.INDEXES	Index specifications created for nicknames.
Type mappings	SYSCAT.TYPEMAPPINGS	User-defined type mappings used in nickname registration and transparent DDL. Default built-in type mappings are not stored in these catalog views. Mapping direction = 'F'/'R'.
Function templates	SYSCAT.FUNCTIONS SYSCAT.ROUTINES	Registered user-defined functions. In V8, SYSCAT.ROUTINES supersedes SYSCAT.FUNCTIONS in V8, but SYSCAT.FUNCTIONS still exists, not documented.
Function mappings	SYSCAT.FUNCMAAPPINGS SYSCAT.FUNCMAPOPTIONS SYSCAT.FUNCMAAPPARMOPTIONS SYSCAT.ROUTINES	User-defined function mappings to map a local function to a remote function.
Passthru privileges	SYSCAT.PASSTHURAUTH	Authorization to allow users to query a specific server using PASSTHRU.
Informational constraints	SYSCAT.TABCONST	Specifies informational constraints associated with nicknames.

Attention: All SYSCAT views are now read only. In order to update information the SYSSTAT view should be used. These views can be updated with SQL UPDATE statements to update the statistics for nicknames and nickname index specifications.

The updateable global catalog views are:

- ▶ SYSSTAT.COLUMNS
- ▶ SYSSTAT.INDEXES
- ▶ SYSSTAT.ROUTINES
- ▶ SYSSTAT.TABLES

This information is collected when the federated system is configured as discussed in 1.4.3, “Configuring the federated system” on page 18. This information can be queried by issuing queries against the catalog.

The DB2 query optimizer uses the information in the global catalog and the data source wrapper to plan the optimal way to process SQL statements. Execution plans for federated queries are chosen by the same DB2 optimizer that optimizes regular queries; the difference is that the federated engine uses the native client interface to each target data source, and sends queries to it in its own dialect.

1.4.3 Configuring the federated system

The DB2 federated server allows you to access and join data from relational and non-relational data sources. By setting the database manager configuration parameter `FEDERATED` to `YES`, the DB2 instance (without DB2 II) allows federated access to other DB2 sources, Informix, and any OLE DB source.

Attention: If you need access to other non-relational or non-IBM relational sources such as Oracle, Sybase, Teradata, or Microsoft SQL databases as well as generic ODBC access, then you need to install DB2 II.

Figure 1-5 on page 19 highlights the basic steps involved in configuring the federated system. Some of these steps may be optional, depending upon the data source being configured. Most of the steps to configure access to a data source can be accomplished through the DB2 Control Center. Use the DB2 Command Center for the steps that require a command line.

Attention: Before configuring access to a data source, ensure that the federated server has been set up properly. It is especially important to:

- ▶ Link the wrapper libraries to the client software (UNIX only).
- ▶ Set up the data source environment variables.

For further details, refer to the *IBM DB2 Information Integrator: Installation Guide for Linux, UNIX, and Windows Version 8.2*, GC18-7036-01.

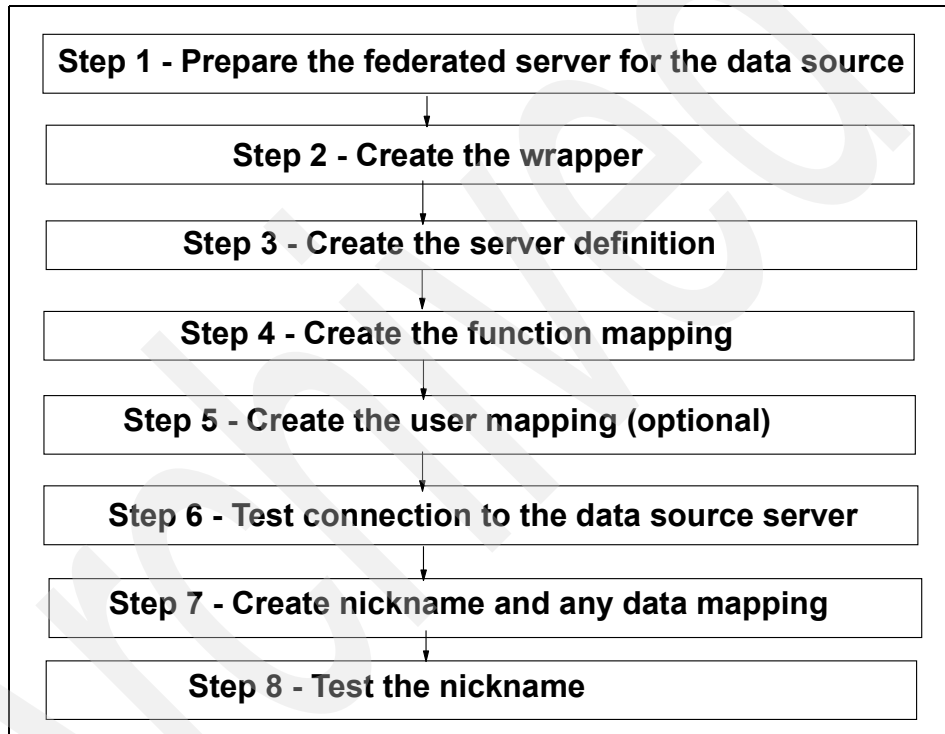


Figure 1-5 Basic steps in configuring a federated system

Each of these steps is described briefly:

1. **Step 1** involves preparing the federated server for the data source. For the DB2 family, this involves cataloging the node and the remote database. For Informix, Sybase, Microsoft SQL Server, Oracle and Teradata data sources, it involves setting up the appropriate client configuration file (for example, Oracle *tnsnames.ora*, Sybase interfaces, SQL Server *odbc.ini*, Informix *sqlhosts*, and Teradata */etc/hosts*) and testing the connection to the data source to be sure the connection is configured correctly before being used with DB2 II.

2. **Step 2** involves creating the wrappers in the federated server. One wrapper is created for each type of data source to be accessed. When a wrapper is created, it is registered in the federated database and the wrappers can now be used to access objects from these data sources.

Wrapper options are used to configure the wrapper or to define how the federated server uses the wrapper. Wrapper options can be set when you create or alter the wrapper. All relational and non-relational data sources use the DB2_FENCED wrapper option. The ODBC data source uses the MODULE wrapper option, and the Entrez data source uses the EMAIL wrapper option. For more details about wrapper options refer to *IBM DB2 Information Integrator Federated Systems Guide Version 8.2*, SC18-7364-01

3. **Step 3** involves creating the server definition that defines the data source to be accessed by the federated database. The name of the data source and other information is part of the server definition.
 - For a relational DBMS (RDBMS), it includes the type and version of the RDBMS, the database name for the data source on the RDBMS, and metadata that is specific to the RDBMS. A DB2 data source can have multiple databases, and therefore a database name is required to identify it as the target. An Oracle data source, on the other hand, can only have a single database, and a database name is therefore not included in the federated server definition of an Oracle data source.
 - For non-relational data sources, you must register a server object because the hierarchy of federated objects requires that specific files that you want to access must be associated with a server object.

During the creation (or alteration) of a server definition of a relational data source, server options can be used to set server attributes that contain information about the data source location, connection security, and some server characteristics that affect performance. These characteristics and restrictions are used by the query compiler in planning the query.

CPU_RATIO, COMM_RATE, IO_RATIO, COLLATING_SEQUENCE, PUSHDOWN, and DB2_MAXIMAL_PUSHDOWN are a few of the server options. Some of the server options are available for all data sources, and others are data source specific. For more details about server options refer to *IBM DB2 Information Integrator Federated Systems Guide Version 8.2*, SC18-7364-01.

Attention: Server options are generally set to persist over successive connections to the data source; however, they can be set or overridden for the duration of a single connection.

The federated system provides the SET SERVER OPTION statement for you to use when you want a server option setting to remain in effect while your application is connected to the federated server. When the connection ends, the previous server option setting is reinstated. The SET SERVER OPTION has no effect on the IUD_APP_SVPT_ENFORCE option with static SQL statements.

4. **Step 4** involves creating functional mappings if the default mappings provided by DB2 II are inadequate and inhibit pushdown of the function to the data source.

DB2 Information Integrator supplies default function mappings between existing built-in relational data source functions, and their built-in DB2 counterpart functions. These default function mappings are implemented in the wrappers.

There are several reasons for creating function mappings, as follows:

- No DB2 function corresponding to a remote data source function is available.
- A corresponding DB2 function is available, but with a specification that is different from that of its remote counterpart.
- A new built-in function becomes available at the data source.
- A new user-defined function becomes available at the data source.

The DB2 catalog view for function mappings is SYSCAT.FUNC mappings.

Function mappings are one of several inputs to the pushdown analysis performed by the query optimizer. If your query includes a function or operation, the optimizer evaluates if this function can be sent to the data source for processing. If the data source has the corresponding function available, then the processing of this function can be pushed down to help improve performance.

A DB2 function template can be used to force the federated server to invoke a data source function. Function templates do not have executable code, but they can be the object of a function mapping. After creating a DB2 function template, you need to create the actual function mapping between the template and the corresponding data source function. The function template allows DB2 II to accept the function name in SQL statements from users and applications. Without the function template, DB2 II would return a syntax error for an unsupported function name. The function mapping tells DB2 II that the

function is supported at a data source and what function to specify in the SQL statement that is to be sent to the data source.

The CREATE FUNCTION MAPPING statement gives considerable control over the scope of the mapping. For example, you can:

- Create a function mapping for all data sources of a specific type, such as all Informix data sources.
- Create a function mapping for all data sources of a specific type and version, such as all Oracle 9 data sources.
- Create a function mapping for all data source objects located on a specific server.
- Disable a default function mapping; default function mappings cannot be dropped.

For further details on function mappings, refer to the *IBM DB2 Information Integrator Data Source Configuration Guide Version 8*, available as softcopy from the Web site:

ftp://ftp.software.ibm.com/ps/products/db2/info/vr82/pdf/en_US/i1y1se81.pdf

5. **Step 5** is an optional step and involves establishing a mapping between the federated server user ID/password and the user ID/password of the data source.

Note: If this mapping is not provided, the wrapper will use the user ID/password information provided at DB2 connect time and pass it to the data source.

If this mapping is provided, this association is called a user mapping and is used by the federated server to successfully connect to the target data source. This association may be created for each user ID that will be using the federated system to send distributed requests.

Note: Each user ID accessing this nickname on DB2 II may be mapped to the remote data source user ID.

6. **Step 6** involves checking to see whether the federated system can connect to the target data source. A passthru session allows you to send SQL statements directly to a data source. Ensure that proper privileges are granted to those users who can use the passthru session for this new data source. For example, with DB2 UDB for z/OS and OS/390®, you can establish a passthru session and issue an SQL SELECT statement on the DB2 system table as follows:

```
SET PASSTHRU servername
```

```
SELECT count(*) FROM sysibm.systables  
SET PASSTHRU RESET
```

In the above example, it is the SELECT statement that causes DB2 II to use the information in the Server and User Mapping definitions to connect under-the-covers to the data source. If the connection is successful, then DB2 II sends a SELECT statement to the data source. However, if the response to the SELECT statement entered by the user is an error message related to the connection attempt, this indicates that there is something wrong with the data source client's connectivity to the data source, or with the information that was provided in either the Server definition or the User Mapping.

7. **Step 7** involves creating a nickname, which is an identifier that is used to reference an object located at the data source that you want to access. The objects that nicknames identify are referred to as data source objects. Nicknames are *not* alternative names for data source objects in the same way that aliases are alternative names. They are pointers by which the federated server references these objects. Nicknames are defined with the CREATE NICKNAME statement.

Additional metadata information can be supplied about the nicknamed object via column options. Data mappings may also be altered or defined between the target data source and DB2 data types in the federated server, if the default mappings provided in the wrappers are inadequate.

Determine whether additional data type mappings need to be defined if you are connecting to a relational data source. Specifying additional data type mappings is necessary if you want to change the default mapping between a DB2 data type and a remote data type. Alternative type mappings, once created with the CREATE TYPE MAPPING statement, can be viewed in the catalog view SYSCAT.TYPEMAPPINGS.

– Data type mappings

Data types of remote data sources must correspond to DB2 data types. An appropriate mapping enables the federated server to retrieve data from the data source. These default data mappings are implemented in the wrappers. DB2 Information Integrator supplies a set of default data type mappings such as the following:

- Oracle type FLOAT maps by default to the DB2 type DOUBLE.
- Oracle type DATE maps by default to the DB2 type TIMESTAMP.
- DB2 UDB for z/OS type DATE maps by default to the DB2 type DATE.

If you want to customize the default mapping provided by DB2 II, then you need to create alternative data type mappings.

In order to use an alternative data type mapping for a nickname, you must create this mapping prior to creating the nickname. If you create the nickname first, you may set the appropriate mapping later, as follows:

- Alter the nickname.
- Change the default mapping types and recreate the nickname.

For further details on data mappings, refer to the *IBM DB2 Information Integrator Data Source Configuration Guide Version 8*, available as softcopy from the Web site:

ftp://ftp.software.ibm.com/ps/products/db2/info/vr82/pdf/en_US/i1y1se81.pdf

For further details on column functions and data mapping, refer to *IBM DB2 Information Integrator Federated Systems Guide*, SC18-7364-01.

8. **Step 8** involves checking to ensure that the nickname has been configured correctly by issuing an SQL statement against it, as follows:

```
SELECT count(*) FROM nickname
```

Note: This query may not be appropriate for some data sources, for example, Lotus Extended Search.

1.4.4 Performance considerations

Probably the most significant concern about federated technology is the issue of acceptable performance. IBM invests heavily in global query optimization research and development in order to improve this area.

The number one factor for good performance in a federated system is the amount of data transferred, and number of interactions between the federated server and data source. The two items that influence this interaction between federated server and data source are the quality of the access plan and the placement of the data. To help with global optimization, the DB2 Information Integrator optimizer takes into account standard statistics from source data (such as cardinality or indexes), data server capability (such as join features or built-in functions), data server capacity, I/O capacity, and network speed.

The following capabilities of the DB2 optimizer have a significant impact on the quality of the access plan generated:

- **Query rewrite** logic rewrites queries for more efficient processing. For example, it can convert a join of unions, which drives a tremendous amount of data traffic, into a union of joins, which leverages query power at the data server and thereby minimizes data traffic back to the federated server. The database administrator (DBA) can define materialized query tables (MQTs),

which the DB2 optimizer can transparently leverage via query rewrite to satisfy user queries.

- ▶ **Pushdown analysis** (PDA) capability identifies which operations can be executed at the data server prior to returning results to the federated server. The amount of processing and filtering that can be pushed down to the remote data sources is key to reducing the volume of data moved between the federated server and the data source. If the remote source supports filtering predicates that are specified in the `WHERE` clause, then the federated server pushes down these predicates to the remote server in order to reduce the amount of data that needs to be shipped back.
- ▶ **Quality of the statistics** is key to enabling the global optimizer to derive an optimal access plan. The statistics need to accurately reflect the current state of the remote object to help the optimizer make the correct cost-based decisions.

Note: The federated server *never* moves data between remote data sources—only between each remote data source and itself.

The actual placement of data in a federated system can also help reduce the interaction between federated server and data source. A reducing join between two tables at the same data source is likely to result in better performance than if the two tables are at different sources. With co-location of the tables, the join can be pushed down and executed at that single source, thus reducing the amount of data and number of interactions with that source. In a federated environment, co-location of tables may be just a consequence of fact, or may be achieved using MQTs and cache tables.

DB2 Information Integrator has some efficient techniques for performing federated joins, as follows:

- ▶ Nested loop join in which the results of SQL sent to one data source are supplied as values for host variables sent in SQL to the second data source
- ▶ Use of hash-joins and merge joins to obtain a join result from two data sources

Attention: Chapter 3, “Key performance drivers of DB2 II V8.2” on page 41, identifies the key DB2 II performance drivers, discusses the pros and cons of each driver, and provides best practices guidelines for their use.

1.5 DB2 Information Integrator topology considerations

Organizations choose Information Technology (IT) platforms and topologies based on their application characteristics, scalability, and availability requirements. For example, for an online stock trading Web application, an organization might choose the AIX platform for the WebSphere Application Server (WAS) and implement both vertical and horizontal WAS clones to achieve the desired scalability and availability requirements of the application.

The key decisions to be made for a DB2 II implementation are:

- ▶ What platform to choose: UNIX, Windows, or Linux?
- ▶ Dedicated federated server or variations of collocation with data sources?
- ▶ What are the capacity requirements of DB2 II?

In this section, we briefly describe the basic topologies and discuss the key criteria involved in choosing between them.

Figure 1-6 on page 27 shows the two basic topologies involved, as follows:

- ▶ The dedicated federated server has no data sources in the same machine as the federated server whatsoever.
- ▶ There are two variations of the collocated federated server, as shown in Figure 1-6 on page 27:
 - The DB2 data source is enabled to be the federated server. There may or may not be other DB2 and non-DB2 relational data sources on the same machine. The machine may also house non-relational data that may be referenced in one or more federated queries.
 - The federated server is in a separate DB2 instance/database on a machine that has DB2 or non-DB2 relational data sources sharing the machine. The machine may also house non-relational data that may be referenced in one or more federated queries.

In general, the DB2 data source enabled to be the federated server option performs better than the option where the federated server has its own DB2 instance/database, especially when the database is partitioned (DPF). However, the federated server in its own DB2 instance/database provides better fault tolerance and isolation.

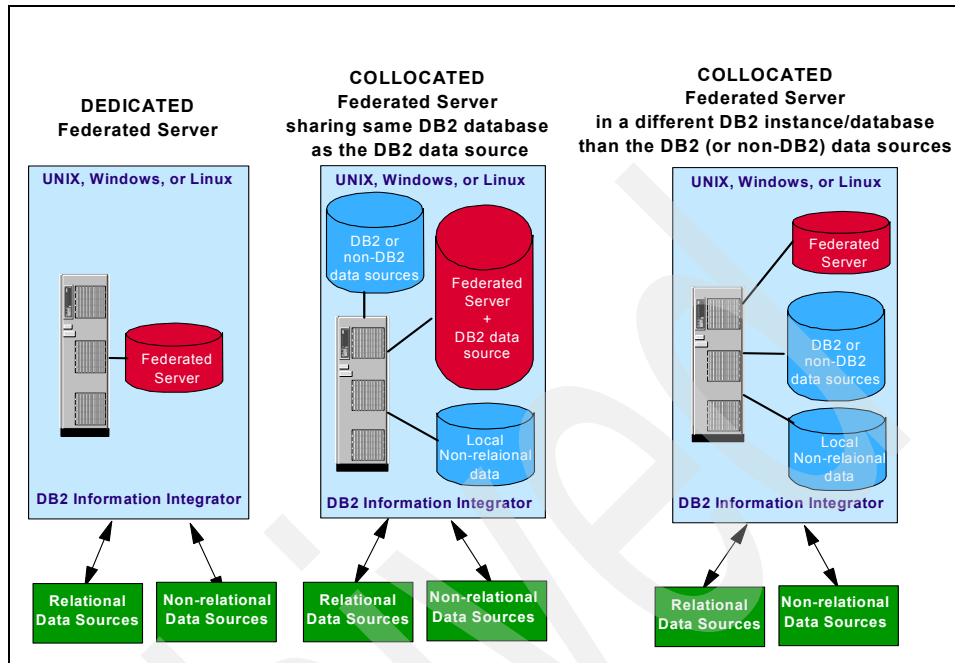


Figure 1-6 DB2 II topologies

The key criteria in choosing a particular DB2 II topology are scalability, availability, total cost of ownership, performance and access considerations.

Attention: The choice of a particular platform (UNIX, Windows or Linux) is largely driven by scalability, availability, and total cost of ownership considerations, and will not be discussed here.

Estimating the CPU and memory capacity requirements of a federated server for a new application is also beyond the scope of this book. However, for organizations that currently have DB2 II installed, Chapter 5, "Capacity planning in an existing DB2 II environment" on page 377, documents a procedure for developing custom query utilization profiles that may be used to estimate the capacity requirements of new applications on the same or different federated server.

1.5.1 Dedicated federated server

A dedicated federated server should be considered when one or more of the following considerations apply:

- ▶ Adequate capacity is not available on existing servers to accommodate the resource requirements of DB2 II (refer to *IBM DB2 Information Integrator Installation Guide for Linux, UNIX and Windows Version 8.2*, GC18-7036-01, for DB2 II prerequisites), or the introduction of DB2 II would cause DB2 UDB version/fixpak conflicts. For example, since a Windows server can only support a single DB2 version/fixpak level, any requirement to support other DB2 UDB version/fixpak levels on the same server will require DB2 II to be installed on a separate Windows server to avoid conflicts. This also applies to DB2 II in a UNIX environment since it does not support an alternate fixpak install even though DB2 UDB does so in UNIX.
- ▶ You need to insulate the impact of DB2 II resource consumption on existing application environments. This isolation also provides the DBA with more flexibility in monitoring and tuning a DB2 II environment.
- ▶ Federated queries do not require access to certain non-relational data sources such as flat files (table-structured files), and XML and Microsoft Excel files, since such files must be located on a local or network mapped drive of the server where DB2 II is installed.

Note: Even if access is required to such non-relational data sources, it may be possible to have a dedicated federated server access remote files by either using nicknames on nicknames, or by using third-party software such as an OpenLink ODBC Client.

- ▶ Federated query performance is considered to be acceptable when accessing data from multiple remote data sources.
- ▶ The additional capital equipment, and non DB2 II licensing and administration costs incurred for the dedicated server are acceptable.

1.5.2 Collocated federated server

A collocated federated server should be considered when the considerations unfavorable to a dedicated federated server apply, such as the following:

- ▶ Adequate capacity is available on existing servers to accommodate the resource requirements of DB2 II (refer to *IBM DB2 Information Integrator Installation Guide for Linux, UNIX and Windows Version 8.2*, GC18-7036-01, for DB2 II prerequisites), or the introduction of DB2 II would not cause DB2 UDB version/fixpak conflicts.

- ▶ Federated queries require access to certain non-relational data sources such as flat files (table-structured files), XML and Microsoft Excel files, since such files must be located on a local or network mapped drive of the server where DB2 II is installed.
- ▶ Federated query performance is considered to be unacceptable when accessing data from multiple remote data sources, and therefore the DB2 II federated server needs to be collocated with the data source server that provides maximum performance gain within the constraints of available capacity.

Note: If one of the data sources referenced is a partitioned (uses DPF) DB2 server, then there are significant performance benefits to be gained by enabling the partitioned database to also function as the federated server. Besides the benefits of inter-partition parallelism for federated queries accessing partitioned tables, the computation partition group (CPG) feature provides the potential for inter-partition parallelism exploitation even for queries that do not access partitioned tables. CPG and other major DB2 II V8.2 performance enhancements are described in Appendix A, “DB2 II V8.2 performance enhancements” on page 429.

- ▶ The additional capital equipment, and non DB2 II licensing and administration costs incurred for implementing a dedicated server are unacceptable.

Archived

Introduction to performance management

In this chapter we provide a general overview of performance management concepts. We also describe the high-level tasks that a DBA typically performs to ensure that her DB2 environment is performing adequately and meeting previously agreed upon service-level objectives. We also discuss a possible problem determination methodology for performing performance problem diagnosis of DB2 environments in general, and DB2 Information Integrator (DB2 II) environments in particular.

The topics covered are:

- ▶ Performance management
- ▶ Types of monitoring
- ▶ Problem determination methodology

2.1 Introduction

One of the main objectives of an IT organization is to ensure that its infrastructure delivers the required performance to ensure that business objectives are continuously met in a constantly evolving and changing business environment.

This requires the IT professional to adopt a strategy that is both *proactive* and *reactive* to conditions and events that would tend to adversely impact IT systems.

The *proactive* effort involves a number of tasks, including the following:

- ▶ Capacity planning of IT resources
- ▶ Choosing the most effective IT architecture for the current and anticipated workload
- ▶ Adopting best practices in application design, development, and deployment
- ▶ Performing rigorous regression testing prior to deployment in a production environment
- ▶ Performing routine monitoring of key performance indicators to forestall potential performance problems, as well as gather information for capacity planning

The *reactive* effort involves having a well-defined methodology for identifying the root cause of a problem, and resolving the problem by applying best practices.

In the following sections we introduce the concept of performance management, describe the different types of monitoring available, and discuss a typical methodology for effective performance problem determination in DB2 II environments.

2.2 Performance management

Most contemporary environments range from stand-alone systems to complex combinations of database servers and clients running on multiple platforms. Critical to all these environments is the achievement of adequate performance to meet business requirements. Performance is typically measured in terms of response time, throughput, and availability.

The performance of any system is dependent upon many factors including system hardware and software configuration, number of concurrent users, and the application workload.

Note: Performance management is a complex issue, and can be defined as modifying the system and application environment in order to satisfy previously defined *performance objectives*.

These performance objectives must be:

- ▶ **Realistic** in that they should be achievable given the current state of the technology available. For example, setting sub-second response times to process millions of rows of data is not achievable.
- ▶ **Reasonable** in that while the technology may be available, the business processes may not require stringent performance demands. For example, demanding sub-second response times for analytic reports that need to be studied and analyzed in detail before making a business decision could be considered unreasonable.
- ▶ **Quantifiable** in that the objectives must use quantitative metrics (numbers, ratios, percentages) instead of qualitative metrics (such as very good, average, etc.). An example of quantitative metrics could be that 95 percent of a particular transaction time must have sub-second response time, while a qualitative metric could be that system availability should be very high.
- ▶ **Measurable** in that one has to be able to measure the performance in order to determine conformance or non-conformance with performance objectives. Units of measurement include response time for a given workload, transactions per second, I/O operations, CPU use, or a combination of the above. Setting a performance objective of sub-second response times for a transaction is moot if there is no way of measuring to determine whether this objective is being met.

Important: Without well-defined performance objectives, performance is a hit-or-miss exercise, with no way of delivering on any service level agreements that may be negotiated with users.

Figure 2-1 highlights the performance management cycle.

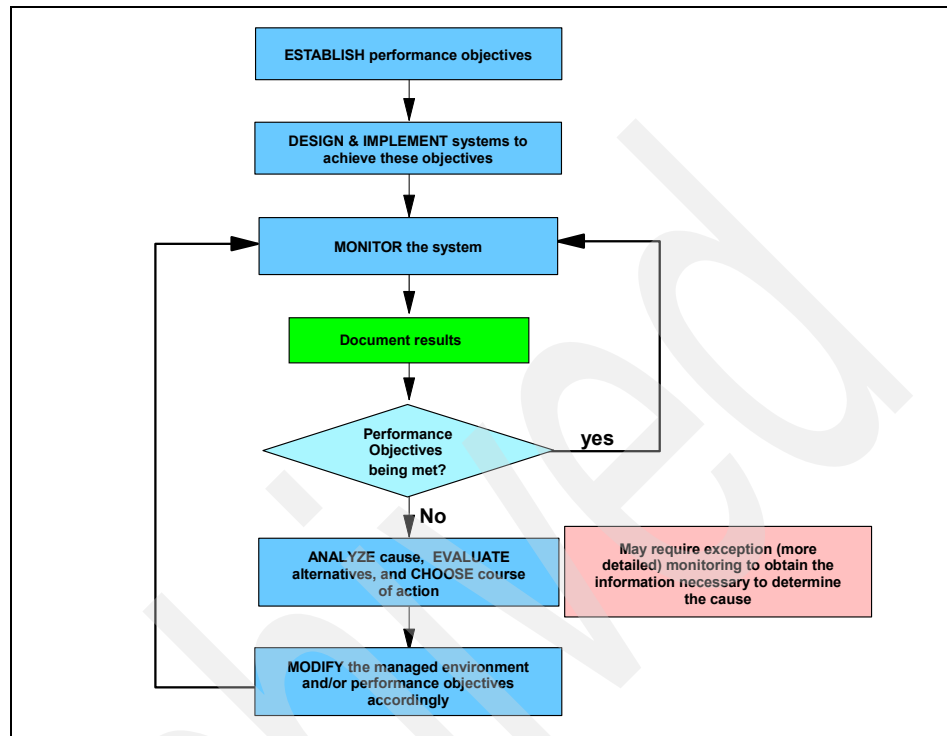


Figure 2-1 Performance management cycle

Performance management is an iterative process that involves constant monitoring to determine whether performance objectives are being met even as environments and workloads change over time. When performance objectives are not being met, then appropriate changes must be made to the hardware and/or software environment, as well as the performance objectives themselves, in order to ensure that they will be met.

From a database perspective, performance problems can arise out of a combination of poor application and system design; inadequate CPU, memory disk, and network resources; and suboptimal tuning of these resources.

Besides using monitoring to determine whether performance objectives are being met, monitoring is also used to:

- Assess the current workload of a system and track its changes over time for capacity planning purposes.

- ▶ Take a proactive approach to performance management by forestalling and resolving potential problems that could impede the achievement of performance objectives.
- ▶ React to unexpected problems by assisting in problem diagnosis.

Attention: In order to meet performance objectives currently and in the future, the DBA needs to develop and execute an appropriate monitoring strategy capable of delivering the required quality of service.

Important: Performance management can only be exercised in controlled environments such as a production system or a regression system.

Remember, “you cannot manage what you cannot control, and you cannot control what you cannot measure”.

Therefore, test and development systems by definition are inherently unmanageable.

2.3 Types of monitoring

There are typically three types of monitoring available to a DBA, as follows:

- ▶ Routine monitoring
- ▶ Online/realtime event monitoring
- ▶ Exception monitoring

Each of these types of monitoring is described briefly in the following sections.

2.3.1 Routine monitoring

The objectives of this type of monitoring are to:

- ▶ Collect information about the workload and stress on the system during normal and peak periods for capacity planning purposes, as well as identify potential performance problems down the road.
- ▶ Ascertain conformance of the system with performance objectives, and record the deviations if any.

The key to this type of monitoring is that it involves analyzing the information collected over a period of time (long history), and then taking corrective action if required to address performance objectives. In other words, there can be a significant delay between information collection and a corrective response. One example of the results of such monitoring is a realization that the number of

transactions has been growing steadily at a 1-percent rate every week, which will necessitate an upgrade of the server in 12 months in order to continue to meet response time objectives.

Another characteristic of such monitoring is the critical need to minimize the overhead it introduces given its requirement to be running constantly or during peak periods.

In some literature, this type of monitoring is further subclassified into continuous monitoring (for normal loads) and periodic monitoring (for peak loads).

From a DB2 perspective, routine monitoring can help identify the root causes of potential performance problems such as:

- ▶ Buffer pool size
- ▶ Dynamic SQL cache size
- ▶ Heap sizes
- ▶ Locklist and maxlocks sizes
- ▶ Application concurrency and isolation issues
- ▶ Disorganized tables
- ▶ Outdated statistics
- ▶ Long running SQL
- ▶ Log and table space utilization

From a DB2 II perspective, routine monitoring can identify root causes of performance problems such as:

- ▶ Missing indexes
- ▶ Missing/stale statistics
- ▶ Incompatible data types in a join
- ▶ Inappropriate DB2 II server (such as DB2_MAXIMAL_PUSHDOWN) or wrapper definitions (such as FENCED)

2.3.2 Online/realtime event monitoring

The objective of this type of monitoring is to be on the lookout for specific events that may either identify a specific problem, or portend problems in the near to immediate future, in order to take prompt corrective action. *Near to immediate future* implies minutes rather than hours.

The key to this type of monitoring is that it involves looking for specific events in a short interval of time (short history) that are known to degrade performance, and having the option to take prompt corrective action to rectify the problem. In other words, there probably needs to be a very short delay between information collection and a corrective response. One example of such an event is the

occurrence of an excessive number of deadlocks in a short period of time, which need to be addressed promptly to ensure that business objectives are not being compromised.

Here, too, the need to minimize the overhead of such monitoring is critical, given that most problems manifest themselves at peak loads.

From a DB2 perspective, online/realtime event monitoring can help identify potential performance problems such as:

- ▶ Deadlocks
- ▶ Long lock waits and time outs
- ▶ Long-running SQL

From a DB2 II perspective, online/realtime event monitoring can help identify potential performance problems such as:

- ▶ Data source server status—alerts if unavailable
- ▶ Relational nickname status—whether valid or invalid

2.3.3 Exception monitoring

This type of monitoring is required when you discover or suspect a problem, and need to identify its root cause in order to apply the appropriate corrective action to fix the problem.

Unlike routine and event monitoring, which are planned occurrences and are designed to have low overheads on the managed system, exception monitoring is driven by problem situations and may impose significant overheads on the managed system. An example of a need for exception monitoring is when the administrator receives a significant number of user complaints about degraded response times, or inability to access the application. The administrator then needs to initiate a series of monitoring actions to home in on the root cause of the problem. This typically involves coming up with a set of hypotheses that could account for the perceived behavior, and then systematically verifying each one in turn until the problem is diagnosed.

From DB2 and DB2 II perspectives, exception monitoring can apply to any of the items identified via routine and online/realtime event monitoring.

2.4 Problem determination methodology

Users may experience performance problems for reasons ranging from:

- ▶ Network connectivity and bandwidth constraints
- ▶ System CPU, I/O, and memory constraints

- Software configuration limitations and constraints
- Poor systems administration skills
- Poor application design
- Poor assumptions about the workload

Given the multitude of possible causes of poor performance, a systematic and consistent approach to problem diagnosis is recommended to ensure prompt and effective resolution of performance problems.

Figure 2-2 describes a typical sequence of steps to be followed when diagnosing performance problems.

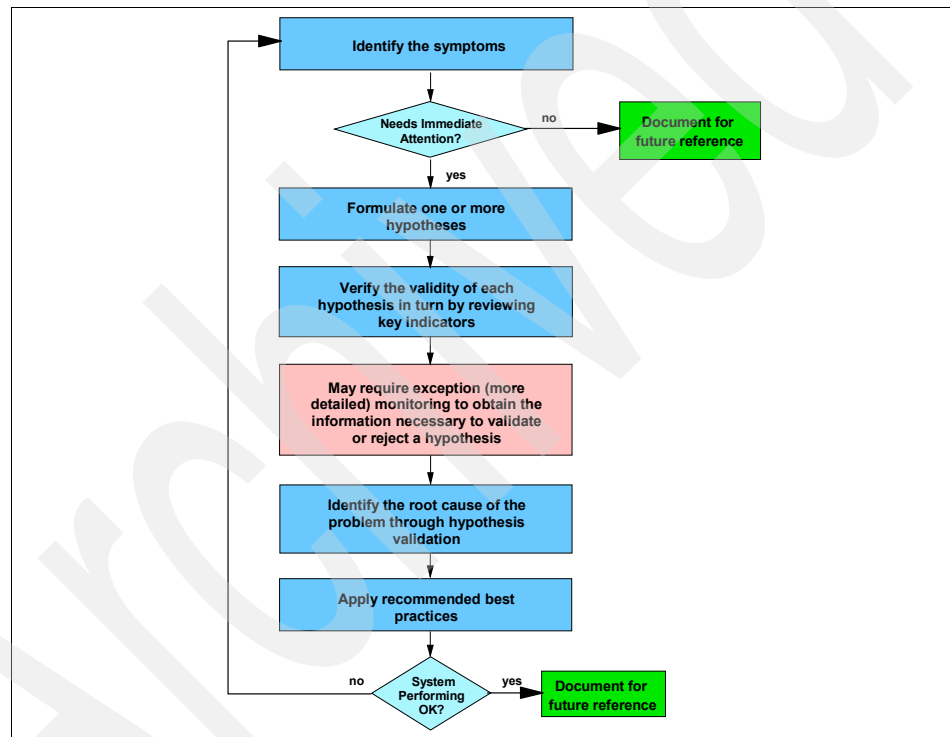


Figure 2-2 A typical problem determination methodology

The entire sequence of steps is triggered by events such as a user complaining about poor response times, error messages appearing on users' screens, alerts or notifications on the DBA console, and alerts in routine monitoring reports.

These symptoms must be evaluated for criticality, as shown by the decision box (“Needs immediate attention”) in Figure 2-2 on page 38.

- ▶ Symptoms that are sporadic and non-disruptive need no immediate action, other than to potentially trigger exception, online/realtime, or additional routine monitoring to gather additional information for possible corrective action in the future.
- ▶ Symptoms that recur frequently and disrupt business processes require prompt attention to avoid adverse business impact. We cover some of these scenarios in this book.
- ▶ Catastrophic events such as a failure of the system, or the application server or database server also need immediate attention such as an immediate restart. These scenarios are *not* discussed in this book.

Based on the symptoms and a knowledge base of prior experiences (both external and internal), one should formulate one or more hypotheses as the potential root cause of the problem. Hypotheses should be formulated that attempt to isolate the cause of the problem to various points in the path of the query through DB2 II to the data source, and the path of the result back to the user. Examining the results of the output of the tests for these hypotheses should focus on where the performance degrades most noticeably between tests on different points in the path.

Each hypothesis should then be tested in turn using all available metrics associated with the application under consideration; this includes system resources, network resources, Web application server resources, and database server resources. Sometimes, the metrics available from routine monitoring and online/realtime event monitoring may be inadequate to validate or reject a particular hypothesis. In such cases, one may have to request additional diagnostic information through more detailed monitoring levels either on the production system itself, or an available comparable regression system. Such monitoring is often referred to as *exception monitoring*.

Once a hypothesis is validated and the root cause problem has been identified, best practices specific to the root cause problem can be applied to attempt to resolve the problem.

Important: Best practices guidelines are based on user experiences for a given workload and environment, and may or may not provide beneficial results in your particular environment. Therefore, a thorough understanding of the fundamentals of the technical architecture and design is required to explore other alternatives, when the documented best practices fail to provide relief.

Problem resolution in such cases tends to be an iterative process, where the application of a best practice may result in the manifestation of new symptoms, and the formulation of a fresh set of hypotheses.

Once the root cause problem has been resolved, the steps executed and the knowledge gained should become part of the knowledge base to assist in resolving future problem situations.

Attention: When applying best practices recommendations, it is vital that changes be implemented *one at a time* and its impact measured before embarking on further changes. Implementing multiple changes simultaneously may make it difficult to assess the impact of a specific change and develop a useful knowledge base for future performance tuning efforts. A knowledge base of all successful and unsuccessful changes in one's environment should be accumulated to develop best practices and recommendations for one's own unique environment.

Chapter 4, “Performance problem determination scenarios” on page 115, describes a performance problem determination methodology for a typical DB2 II environment.

Key performance drivers of DB2 II V8.2

In this chapter we describe the compilation and execution flow of a federated query, and discuss key DB2 II performance drivers, their pros and cons, and best practices recommendations for achieving superior performance.

The topics covered are:

- ▶ Compilation flow of a federated query
- ▶ Execution flow of a federated query
- ▶ Key performance drivers

3.1 Introduction

In a federated system, data sources appear as objects (tables) in a single collective DB2 UDB database to end users, client applications, and developers.

Users and applications interface with the federated database managed by the federated server. The federated database contains a system catalog that contains entries that identify data sources and their characteristics. The federated server consults the information stored in the federated database system catalog and the data source wrapper to determine the best plan for processing SQL statements.

The federated system processes SQL statements as if the data sources were ordinary relational tables or views within the federated database. As a result:

- ▶ The federated system can join relational data with data in non-relational formats. This is true even when the data sources use different SQL dialects, or do not support SQL at all.
- ▶ The characteristics of the federated database take precedence when there are differences between the characteristics of the federated database and the characteristics of the data sources:
 - Suppose the code page used by the federated server is different from the code page used by the data source. Character data from the data source is converted based on the code page used by the federated database, when that data is returned to a federated user.
 - Suppose the collating sequence used by the federated server is different from the collating sequence used by the data source. Any sort operations on character data are performed at the federated server instead of at the data source.

The basic flow of a federated query is as follows:

1. User or application submits a query.
2. The federated server decomposes the query by source.
3. The federated server and wrappers collaborate on a query plan.
4. The federated server implements the plan through query execution.
5. Wrappers communicate with the data sources and send SQL statements through each source's API.
6. Sources return data to the wrappers.
7. Wrappers return the data to the federated server.
8. The federated server compensates for work that the data sources are unable to do and combines data from different sources.

9. The federated server returns the final result set to the user or application.

After a query is submitted, the federated server consults its system catalog to determine the nicknames' underlying data sources and their characteristics, including the wrappers designated to access these data sources. The federated server then devises alternative strategies, called access plans, for evaluating the query. Such a plan might call for parts of the query to be processed by the data sources, by the federated server, or partly by the sources and partly by the federated server. The federated server chooses to execute the plan with the lowest cost. Statistics and index information for nicknames are the primary input to the optimizer for determining the costs of alternative plans and for the ultimate decision of which plan is the lowest cost for execution.

There are two phases of query processing: Query planning and query execution. The query planning phase occurs during compile time, while the query execution phase occurs during runtime.

Figure 3-1 shows the flow of query processing in a federated system during compile time and run time.

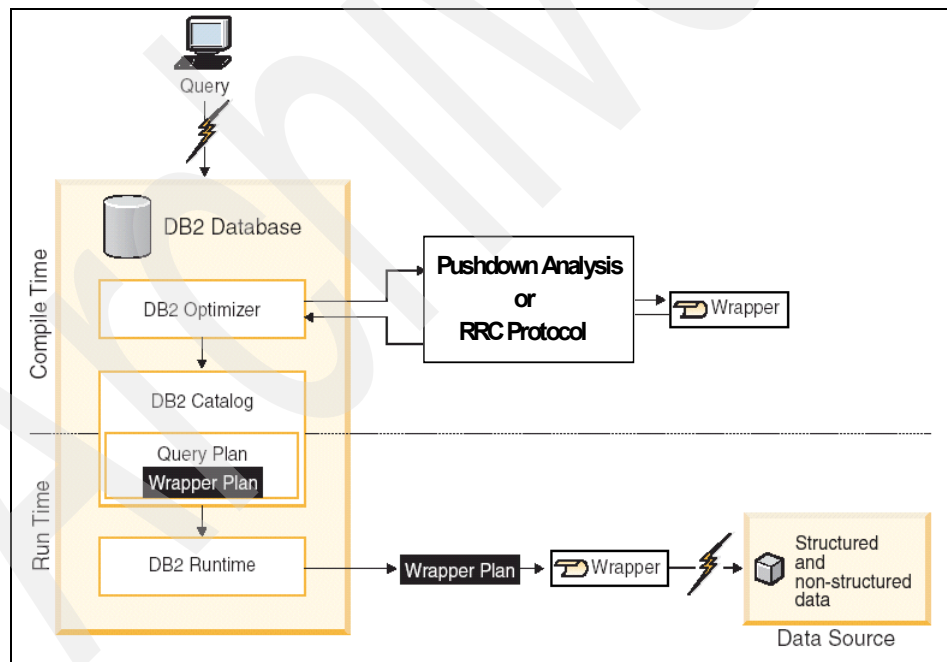


Figure 3-1 Federated query processing flow

Note: The access plan for static SQL statements is created at bind time and contained in a package. The package contains the access plan for all SQL statements in the application program.

For dynamic SQL statements, the access plan for each SQL statement is created when the application is executed. Therefore, compile times are included in the response times experienced by the user.

In the following subsections we describe:

- ▶ Compilation flow of a federated query
- ▶ Execution flow of a federated query
- ▶ Key performance drivers in a federated environment

3.2 Compilation flow of a federated query

Figure 3-2 shows the steps involved in generating an optimal access plan for a federated query, and some of the options that influence the individual steps during compilation.

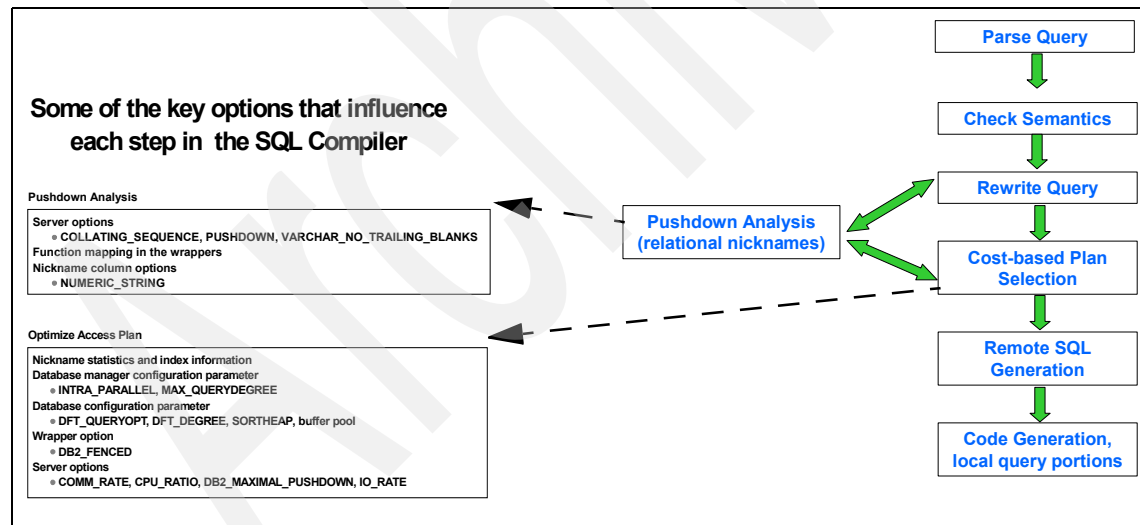


Figure 3-2 SQL Compiler query analysis flow

As part of the SQL Compiler process, the query optimizer analyzes a query. The SQL Compiler develops alternative strategies, called access plans, for processing the query. The access plans might call for the query to be:

- ▶ Processed by the data sources
- ▶ Processed by the federated server
- ▶ Processed partly by the data sources and partly by the federated server

As mentioned earlier, DB2 UDB evaluates the access plans primarily on the basis of information about the data source capabilities and the data obtained from the wrapper and the global catalog. DB2 UDB decomposes the query into segments that are called query fragments. Typically it is more efficient to push down a query fragment to a data source, if the data source can process the fragment. However, the query optimizer takes into account other factors, such as:

- ▶ Amount of data that needs to be processed
- ▶ Processing speed of the data source
- ▶ Amount of data that the fragment will return
- ▶ Communication bandwidth

Note: Pushdown analysis is only performed on relational data sources. Non-relational data sources use the request-reply-compensate protocol.

The query optimizer generates global access plans, which include SQL fragments to access remote data, and executable code to access local DB2 data. In order to maintain an accurate and complete cost model, the optimizer estimates the cost of remote SQL fragments using the statistics available on the nickname. The optimizer estimates the cost of these alternative plans, and chooses the plan it believes will process the query with the least resource cost. If any of the fragments are to be processed by data sources, the federated server submits these fragments to the data sources. After the data sources process the fragments, the results are retrieved and returned to the federated server. If the federated server performed any part of the processing, it combines its results with the results retrieved from the data source. The federated server then returns all results to the client.

Each of the steps in Figure 3-2 on page 44 is described briefly, as follows. The options listed in Figure 3-2 on page 44 are briefly described here; however, they are described in detail in 3.4, “Key performance drivers” on page 54:

1. **Parse Query** involves analyzing the SQL query to check the syntax. When parsing the query is completed, an internal representation of the query called the query graph model is created.
2. **Check Semantics** involves checks to ensure that the query semantics are correct; for example, checks to ensure that the data type of the column

specified for the AVG column function is a numeric data type, determining whether the views in the query need to be merged or materialized.

3. **Rewrite Query** involves the SQL compiler rewriting the query to make it more efficient, while retaining the semantic equivalence of the original query. Categories of query rewrite include the following:

- *Operation merging*, where a subquery can sometimes be merged into the main query as a join, which the optimizer has more choices to determine the most efficient access plan
- *Operation movement*, where the optimizer may remove the DISTINCT operation to reduce the cost of operation
- *Predicate translation*, where the SQL compiler may rewrite queries to translate existing predicates to more optimal predicates

4. **Pushdown Analysis** (PDA) is bypassed unless it is a federated database query.

The primary task of pushdown analysis is to determine whether some or all parts of a query in the optimized SQL can be “pushed down”, that is, processed at the remote source(s). The ability to push down depends on the availability of the needed functionality at the remote source and on the server options set such as COLLATING_SEQUENCE. A secondary task of pushdown analysis is to attempt to transform the query into a form that can be better optimized by both the DB2 optimizer and remote query optimizers.

Query rewrite invokes PDA to decide whether to enable certain traditional rewrite for federated queries. Query rewrite also performs specific rewrite for federated queries such as outer join reordering for maximum pushdownability, and function template in select list pushdown.

The following DB2 II options impact the operations that can be pushed down:

- Server options COLLATING_SEQUENCE, PUSHDOWN, and VARCHAR_NO_TRAILING_BLANKS
 - Setting the COLLATING_SEQUENCE server option to “Y” tells the federated database that the remote data source collating sequence matches the DB2 collating sequence. Such a setting allows the optimizer to consider pushing down order-dependent processing at a data source for CHAR and VARCHAR columns, which can improve performance. For other types of columns (DATE, TIME, TIMESTAMP, INTEGER, FLOAT, DECIMAL), the optimizer always has the option to consider pushing down order processing.
 - Setting the PUSHDOWN server option to 'Y' (the default) lets DB2 UDB consider letting the remote data source evaluate operations. If PUSHDOWN = 'N', DB2 will send to the remote data source SQL statements that include only SELECT with a column list. Equality

predicates (such as WHERE COL1 =), column and scalar functions (such as MAX and MIN), sorts (such as ORDER BY or GROUP BY), and joins will *not* be included in any SQL sent to the data source, thereby having a negative performance impact.

- The VARCHAR_NO_TRAILING_BLANKS server option is for varying-length character strings that contain no trailing blanks. The SQL Compiler will factor in this setting when it checks for all operations (such as comparison operations) performed on columns.
- Default function mappings are built into the data source wrappers. The federated database will compensate for functions that are not supported by a data source. When compensation occurs, that processing occurs at the federated server.
- The NUMERIC_STRING nickname column option applies to character type columns (CHAR and VARCHAR). It should only be set for those character columns that contain only numeric values because numbers are always sorted the same, regardless of collating sequence. On setting this option (to 'Y'), the optimizer has the choice of performing the sort either locally or remotely (even though the collating sequences differ), which may lead to an improved access plan and reduce execution time.

Note: The VARCHAR_NO_TRAILING_BLANKS server option is also available as a nickname column option, which gives the DBA the flexibility to identify specific Oracle columns that contain no trailing blanks.

Important: If pushdown analysis determines that an operation cannot or should not be performed at the data source, the optimizer will only be able to evaluate plans that perform the operation at the federated server. If pushdown analysis determines that an operation can be performed at the data source, the optimizer is allowed to evaluate plans that perform the operation at the data source and plans that perform the operation at the federated server. The optimizer will then estimate the cost of the alternative plans to determine where the operation will be performed based on cost.

5. **Cost-based Plan Selection** involves creating an executable access plan, or section, for the query. Information about the access plan for static SQL is stored in the system catalog tables.

Based on this analysis, the query optimizer evaluates the alternatives and chooses the access plan based on cost. Even though an operation can be pushed down, it does not mean that it will be, because the optimizer might

choose to not perform an operation directly on a remote data source because it is more costly.

The following are some of the key options that impact the selection of an optimal access path:

- Nickname statistics and index information
Defining appropriate indexes and keeping the database objects' statistics up-to-date is critical to the DB2 optimizer choosing the optimal access.
- Database manager configuration parameters
 - Setting INTRA_PARALLEL to “YES” (the default is “NO”) allows DB2 to use intra-partition parallelism for federated queries that access local tables on the federated server.
 - MAX_QUERYDEGREE specifies the maximum degree of intra-partition parallelism that is used for any SQL statement executing on this instance of the database manager. An SQL statement will not use more than this number of parallel operations within a partition when the statement is executed. The INTRA_PARALLEL database manager configuration parameter must be set to “YES” to enable the database partition to use intra-partition parallelism. The default value for this configuration parameter is -1, which means that the system uses the degree of parallelism determined by the optimizer; otherwise, the user-specified value is used.
- Database configuration parameters
 - DFT_QUERYOPT is used to direct the optimizer to use different degrees of optimization when compiling SQL queries. This parameter provides additional flexibility by setting the default query optimization class used when neither the SET CURRENT QUERY OPTIMIZATION statement nor the QUERYOPT option on the bind command are used. The default level of 5 should normally not be modified.
 - DFT_DEGREE specifies the default value for the CURRENT DEGREE special register and the DEGREE bind option. The default value is 1, which means no intra-partition parallelism. A value of -1 means the optimizer determines the degree of intra-partition parallelism based on the number of processors and the type of query. The degree of intra-partition parallelism for an SQL statement is specified at statement compilation time using the CURRENT DEGREE special register or the DEGREE bind option. The maximum runtime degree of intra-partition parallelism for an active application is specified using the SET RUNTIME DEGREE command.
 - SORTHEAP defines the maximum number of private memory pages to be used for private sorts, or the maximum number of shared memory pages to be used for shared sorts. The size of the sort heap can impact

sort performance (causing sort overflows) or the selection of a hash join as an optimal access path. SORTHEAP is a factor when there will be large intermediate result sets at the federated server. A large SORTHEAP allows the intermediate result set to be contained in memory. A small SORTHEAP will cause large intermediate result sets to overflow from sortheap to the buffer pool for the temporary table space. When increasing the value of SORTHEAP, you should examine whether the SHEAPTHRES database manager configuration parameter also needs to be adjusted.

- Buffer pool associated with the temporary table space for sort can impact sort performance by minimizing page overflows to disk, and the buffer pool size is considered in estimating the costs of a sort that overflows to it. If there are large temp tables (intermediate result sets) or large sorts that overflow from a sortheap allocation, then a large buffer pool for the temporary table space will result in these temp tables and sorts being held in memory, thereby avoiding disk I/O to the file containers of the temporary tablespace. The buffer pool associated with local tables in the federated server should also be tuned to affect optimal access path selection.
- Wrapper option
 - DB2_FENCED specifies whether the wrapper runs in fenced or trusted mode; the default is DB2_FENCED = “N”, which is trusted. Setting the DB2_FENCED option to “Y” (fenced mode) enables the optimizer to use inter-partition parallelism for nickname data on a federated server. In fenced mode operation, nickname data can be processed at various database partitions in parallel, thereby improving query performance.
- Server options
 - DB2_MAXIMAL_PUSHDOWN specifies whether the primary criteria that the query optimizer uses when choosing an access plan should be based on cost or based on the user requirement that as much query processing as possible be performed by the remote data sources. For instance, there can be cases where a plan with more interactions with a data source takes better advantage of indexes at the data source or enables faster join techniques (such as hash join) at the federated server.

Setting DB2_MAXIMAL_PUSHDOWN = “Y” (default is “N”) causes reduced network traffic to become the overriding criteria for the query optimizer. The query optimizer uses the access plan that has the fewest number of SHIP operators. Setting this server option to “Y” forces the federated server to use an access plan that might not be the lowest cost plan.

- **CPU_RATIO** and **IO_RATIO** specify how much faster or slower the data source CPU and IO_RATIO speed is compared with the federated server CPU speed and I/O rates, respectively. A low ratio indicates that the data source workstation CPU (I/O) is faster than the federated server workstation CPU (I/O). For low ratios, the optimizer will consider pushing down operations that are CPU (I/O) intensive to the data source. A low ratio is a value that is less than 1.
 - **COMM_RATE** specifies the speed of the communication network between the data source and the federated server. A low communication rate indicates slow network communication between the federated server and the data source. Lower communication rates encourage the query optimizer to reduce the number of messages and amount of data sent to or from this data source. If the **COMM_RATE** server option is set to a very small number, the optimizer produces a query requiring minimal network traffic.
6. **Remote SQL Generation** relates to a set of steps that generate efficient SQL statements based on the SQL dialect of the data source.
 7. **Code Generation (local query portions)** is the final step during which the compiler uses the access plan and the query graph model to create an executable access plan, or section, for the local query portions. Information about access plans for static SQL is stored in the system catalog tables. When the package is executed, the database manager will use the information stored in the system catalog tables to determine how to access the data and provide results for the query.

3.3 Execution flow of a federated query

Very simply, the federated server distributes the query fragment assigned to each data source to the corresponding wrappers, which in turn submits the query fragment to the data source and retrieves the results. These steps are typical, and can vary depending on how a specific source handles the concept of a connection, whether the source accepts requests via a query language or through some other API, what kind of result set cursors or iterators (if any) are supported by a source, and so forth.

The typical process of distributing a query, executing it, and returning its results is as follows:

1. The federated server passes authorization information to the wrapper and requests it to establish a connection to the data source.

2. The wrapper establishes the connection requested, and submits the query fragment to the referenced data source. The wrapper then obtains an iterator for the result set that the wrapper is to retrieve.

Note: The federated server will reuse an existing connection if there already exists an established connection within the same application to the data source.

3. The federated server requests a row of results from the wrapper, which in effect “forwards” the request to the data source.
4. The data source executes a portion of a query fragment in order to return the requested row.
5. The wrapper retrieves the requested row, converts the types of the data in the row to the federated server data types, and copies the converted types into buffers. Results from data sources are retrieved in multi-row blocks if the isolation level and parameters of the application permit this. So, results can be retrieved from the data source to the federated server in blocks, and DB2 II can send results to the application in multi-row blocks. The block size for both is determined by the DB2 II database manager configuration parameter RQRIOBLK. The default size is 32,767 bytes. The maximum size allowed is 65,535 bytes.
6. The federated server reads the data from the buffers and processes the data.
7. The federated server, the data source, and the wrapper repeat the three previous steps for each successive row of results.
8. The wrapper retrieves the last row of the result set from the data source and indicates this to the federated server.
9. If the same application references the same data source again, then the connection to the data source will be reused. Otherwise, the connection to the data source will be terminated when the application terminates its connection to the federated server, or 100 commits have been processed without reference to this data source.
10. The wrapper disconnects from the data source.

Figure 3-3 on page 52 is a pictorial representation of these steps, and highlights some of the key options that influence performance in the execution flow.

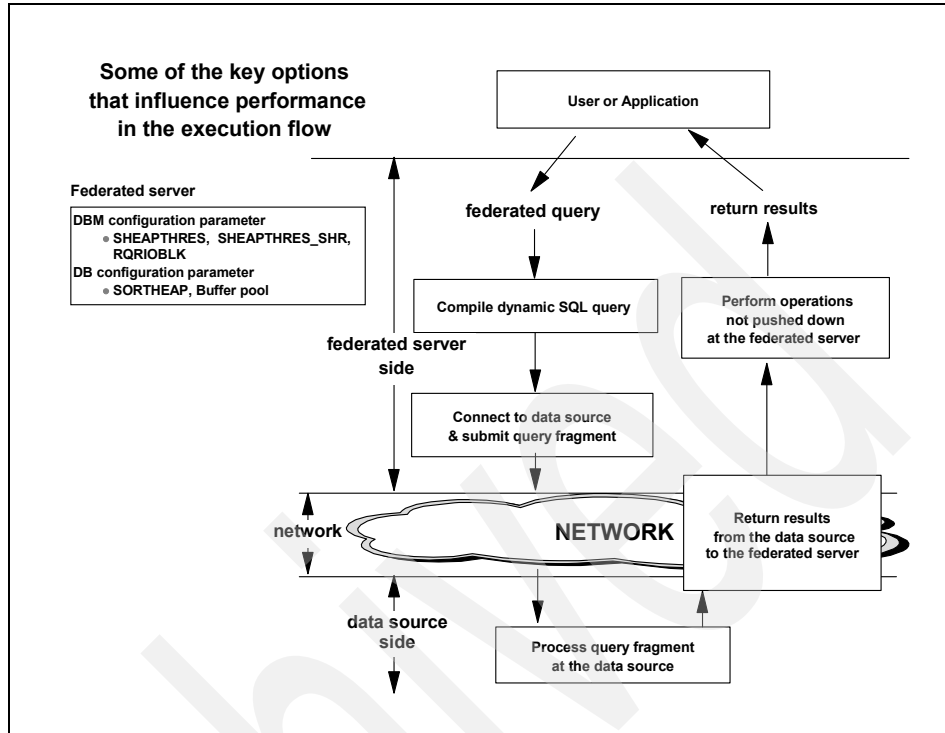


Figure 3-3 Query execution flow

Figure 3-3 shows that a user or application may submit a query to the federated database that is then executed as follows:

- 1. Compile dynamic SQL query.**

If the query is dynamic SQL then it needs to be compiled by the SQL Compiler before execution, as discussed in 3.2, “Compilation flow of a federated query” on page 44. Once the access plan has been generated, it can be executed.

- 2. Connect to the data source and submit query fragment.**

The federated server connects to the data source(s) involved using the API of the source’s client software. The federated server’s wrapper, server, and user mapping options like the node name, database name, remote_authid and remote_password are required to establish this connection.

- 3. Process query fragment at the data source.**

Once the query fragment is passed to the data source, it is executed as any other query at that data source. This means that the data source optimizes the query fragment using its own optimizer and then executes the query.

The query text submitted to the data source can be found in the RMTQTX object for the SHIP operator in the **db2exfmt** output for the user query, as shown in Example 4-9 on page 138. The federated server has very limited control over how the data source processes the query.

DB2 II provides metrics (**Total execution time (sec.ms)** and **Rows read** fields in the dynamic cache snapshot for a remote query fragment, as shown in Example 4-8 on page 136), that provide the total time spent executing the query fragment at the data source, and the number of rows returned to the federated server from the data source. This information can then be used to direct the investigation of performance problems to the data source administrator.

4. Return results from the data source to the federated server.

This activity is shown as spanning the federated server, network, and data source. As already mentioned, the wrapper retrieves each row requested by the federated server, converts the types of the data in the row to the federated server data types, and copies the converted types into buffers.

Important: The amount of “traffic” between the data source and the federated server tends to have a significant impact on the performance of the query on the federated server.

Traffic corresponds to the number of interactions (roughly equal to the number of requests to the data sources) and the amount of data (in the result set) moved.

Data only moves from the data source to the federated server, never the other way around. For example, in a query involving a two-way join with data in two different remote sources, both sources send the data *to* the federated server, and the join occurs locally.

The number of requests to the data sources and the amount of data (number of rows and width of each row) transferred is influenced by the degree of pushdown that occurs. The DB2_MAXIMAL_PUSHDOWN server option tends to minimize the number of interactions with the data sources as well as the amount of data transferred.

The DB2 database manager configuration parameter RQRIOLBK controls the blocksize for receiving data from data sources across the network. When large results sets are involved, the time required to transfer this data from the data source to the federated server could be significant and can impact performance.

5. Perform operations not pushed down at the federated server.

Operations not performed at the data source need to be executed at the federated server on the results returned from the data source, before returning results to the user or application.

These operations include predicates that were not pushed down for either cost optimization reasons, unavailability of functionality at the data sources, or joins with local data and other data sources.

The performance of these operations at the federated server is impacted by the SORTHEAP DB2 database configuration parameter, SHEAPTHRES DB2 database manager configuration parameter, and the size of the buffer pool for the temporary tablespace used in sorts. All these parameters affect the runtime performance of the sort process.

The dynamic SQL snapshot shown in Example 4-8 on page 136 provides information about the total execution time spent at the federated server¹ and the amount of time spent in sorting (**Total sort time**) for a given query. It also provides information about buffer pool activity associated with this query.

3.4 Key performance drivers

Probably the most significant concern about federated technology is the issue of acceptable performance.

DB2 II provides a number of capabilities to enhance federated query performance—from the deployment of superior query optimization technologies, to providing the DBA with effective monitoring and tuning facilities to diagnose the performance of poorly performing queries, and a number of configuration options to fine tune individual queries.

The key performance drivers in a federated environment depend upon whether local data is accessed in a federated query in a collocated federated server environment where the federated server shares the same DB2 database as the data source environment (as described in 1.5.2, “Collocated federated server” on page 28).

When the federated environment is collocated, all the performance considerations that apply to a vanilla DB2 environment (without DB2 II) apply in addition to the specific considerations of a dedicated federated server environment, as described in 1.5.1, “Dedicated federated server” on page 28.

¹ Obtained by subtracting the **Total execution time (sec.ms)** for the user-entered query from the sum of all the **Total execution time (sec.ms)** times of all the corresponding query fragments assuming serial access to the data sources.

Table 3-1 provides a very high-level overview of the key performance drivers in a vanilla non-DPF DB2 environment as compared to a DB2 II environment (with and without local data). Refer to the redbook *DB2 UDB ESE V8 non-DPF Performance Guide for High Performance OLTP and BI*, SG24-6432, for a detailed discussion of DB2 performance.

Table 3-1 Key performance drivers - non-DPF DB2 in relation to DB2 II

Key performance drivers	DB2 non-DPF	DB2 II (local data)	DB2 II (no local data)
System environment considerations			
I/O placement considerations <ul style="list-style-type: none"> DB2 logs, regular tablespaces, large tablespaces, temporary tablespaces 	Yes	Yes	Limited
Log considerations <ul style="list-style-type: none"> Archive logging, dual logging, DB2 logs, database configuration parameters logprimary, logsecond, logfilsiz, logbufsz, mincommit, and blk_log_dsk_full 	Yes	Yes	Minimal
Monitor switch settings <ul style="list-style-type: none"> DFT_MON_BUFPOOL, DFT_MON_LOCK, DFT_MON_SORT, DFT_MON_STMT, DFT_MON_TABLE, DFT_MON_TIMESTAMP, DFT_MON_UOW, HEALTH_MON 	Yes	Yes	DFT_MON_BUFPOOL, DFT_MON_STMT, DFT_MON_SORT, DFT_MON_TIMESTAMP, HEALTH_MON
Database Manager configuration and database configuration parameters			
Connection considerations <ul style="list-style-type: none"> Database manager configuration parameters max_connections, maxagents, max_coordagents, maxcagents, num_poolagents, num_initagents; and database configuration parameter maxappls 	Yes	Yes	Yes
Buffer pool considerations (data) <ul style="list-style-type: none"> Number of buffer pools, size of buffer pools, page size of buffer pools, block-based buffer pools, and database configuration parameters chngpgs_thresh, num_iocleaners, num_ioservers 	Yes	Yes	No
Buffer pool considerations (sort)	Yes	Yes	Yes

Key performance drivers	DB2 non-DPF	DB2 II (local data)	DB2 II (no local data)
Locking considerations ► Database configuration parameters locklist, maxlocks, locktimeout, dlchktme	Yes	Yes	Minimal
Package considerations ► Database configuration parameter pckcachesz	Yes	Yes	Yes
Catalog cache considerations ► Database configuration parameter catalogcache_sz	Yes	Yes	Yes
Sort considerations ► Database configuration parameters sortheap, sheapthres, sheapthres_shr	Yes	Yes	Yes
Parallelism considerations ► Database manager configuration parameter intra_parallel, max_querydegree	Yes	Yes	Yes
Various heaps	Yes	Yes	Depends
RQRIOLBK	Yes	Yes	Yes
DB2 II wrapper, server, and column options			
DB2_FENCED wrapper option	N/A	Yes	Yes
COLLATING_SEQUENCE, COMM_RATE, CPU_RATIO, IO_RATIO, PUSHDOWN, DB2_MAXIMAL_PUSHDOWN, VARCHAR_NO_TRAILING_BLANKS server options	N/A	Yes	Yes
Function mapping	N/A	Yes	Yes
Data type mapping, NUMERIC_STRING, VARCHAR_NO_TRAILING_BLANKS	N/A	Yes	Yes
Application-related considerations			
Table/nickname design ► Data type mapping, informational constraints, cache tables, MQTs	Yes	Yes	Yes

Key performance drivers	DB2 non-DPF	DB2 II (local data)	DB2 II (no local data)
Index (specification) design ► Keep them current and synchronized, add index specifications when it can assist the optimizer in making superior decisions	Yes	Yes	Yes
Statistics ► Keep them current and synchronized	Yes	Yes	Yes
Efficient SQL	Yes	Yes	Yes

In this section, we discuss key performance drivers in a dedicated federated server environment, discuss their pros and cons, provide best practices recommendations, and identify the facilities for monitoring them.

This section is organized as follows:

- Performance factors
- Federated server considerations
- Data source considerations
- Writing efficient federated SQL queries
- Hardware and network

3.4.1 Performance factors

Some portions of a federated query execute at the federated server, while other portions execute at one or more remote data sources. The federated query may also access local data residing on the federated server. Therefore, tuning a federated query involves tuning the federated server environment (federated server side), the network, and the remote data source(s) environment (data source side).

Figure 3-4 on page 58 provides an overview of the tuning options available for a federated query at the federated server and remote data source(s). Each of these tuning options will be covered in detail in the following sections.

Attention: While important, we will not cover the network tuning aspects here.

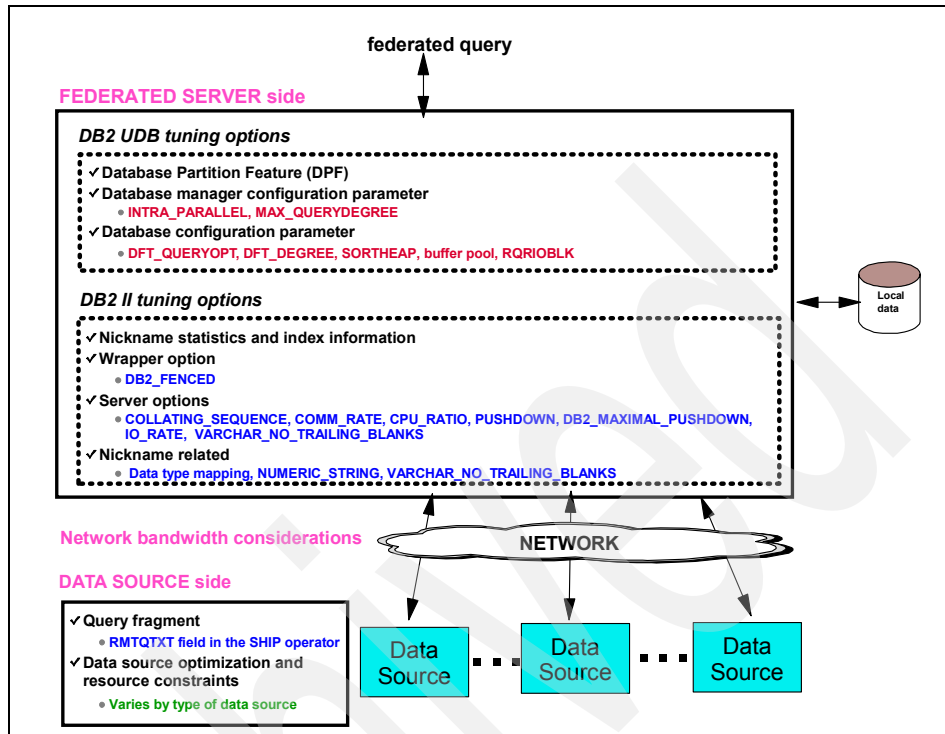


Figure 3-4 Performance considerations overview

Factors that influence federated query performance include:

1. Quality of the generated query execution plans at the federated server. The query execution plan influences the number of interactions required between the federated server and the remote sources, and the amount of data that is moved.
2. The bandwidth latency of the intervening communication network.
3. The processing power of the remote machine(s).
4. Quality of the execution plan(s) at the remote data source(s).
5. How efficiently federated SQL queries are written.
6. The processing power of the local machine.
7. The degree to which the federated server (both the system and database server) is well tuned.

Steps 1, 6, and 7 relate to federated server side performance. Step 2 relates to network bandwidth performance, and steps 3 and 4 relate to remote data source side performance. Step 5 relates to writing efficient federated SQL queries.

Figure 3-5 shows the elements of federated query performance, and organization of topics in this section.

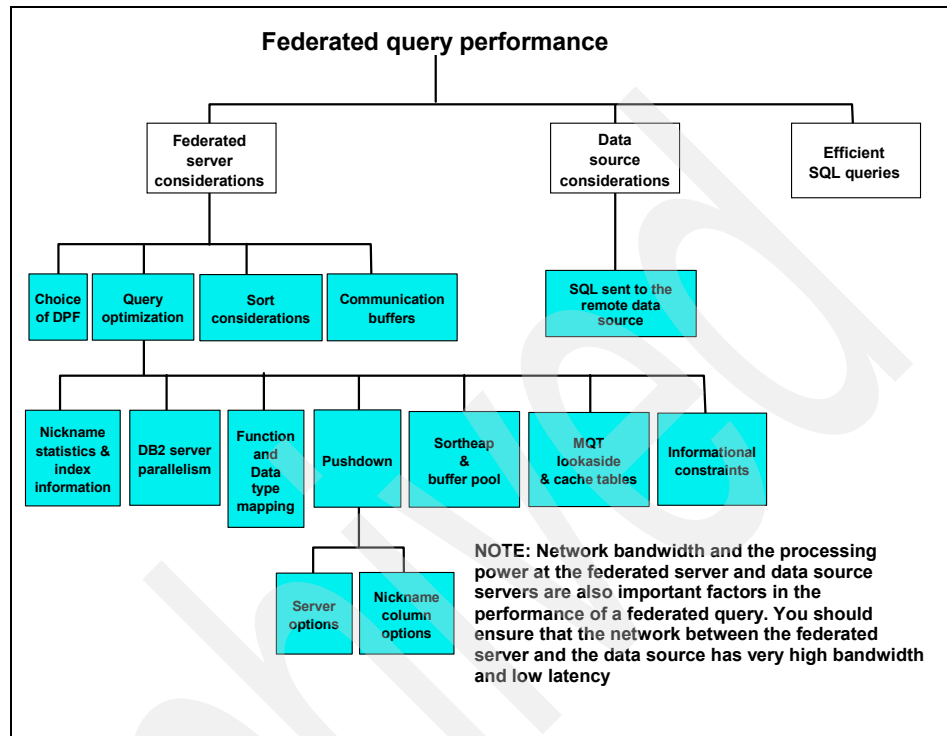


Figure 3-5 Federated query performance elements

Important: While any one or more of the seven factors in the list may be the most significant contributor to a particular federated query's poor performance, in general, the number of interactions with remote data sources and the volume of data (number of rows and width of each row) returned from each data source play a significant role in a federated query's performance.

The amount of data moved (or traffic) mainly depends on three factors:

- The amount of processing and filtering that can be pushed down (see "Pushdown" on page 78) to the remote data sources

If there are some filtering predicates in the WHERE clause, and the remote source is able to apply those predicates, then the federated server pushes down these predicates to the remote server to reduce the amount of data that needs to be shipped back.

- ▶ Data placement among multiple sources

If you join two tables and they are both on the same data source so that the join can be done at that data source without moving the tables out, then that usually results in better performance than if the two tables resided at two different sources.

In a join between tables that are not colocated, data from both tables must be moved to the federated server, which will then do the join.

Note: The federated server *never* moves data between remote data sources—only between each remote data source and itself.

- ▶ Efficiency of the join performed by DB2 II between data sources

Accurate statistics and index information for nicknames are the most important for helping the optimizer pick the most efficient join strategy.

DB2 Information Integrator has some efficient techniques for performing federated joins, as follows:

- ▶ Use of hash joins and merge scans to obtain a join result from two data sources
- ▶ Nested loop join in which the results of SQL sent to one data source are supplied as values for host variables sent in SQL to the second data source

3.4.2 Federated server considerations

Performance at the federated server is impacted by the following:

- ▶ Presence of local DB2 data that must be tuned as in any other DB2 database environment.
- ▶ Processing that was not or could not be pushed down to the remote data sources. This includes:
 - Sorts for joins, aggregations, and ordered sequence of results
 - Applying predicates
 - Compensation

Important: Performance at the federated server can be improved through the choice of DPF as a federated server platform; better optimized queries that exploit pushdown, parallelism, and superior join strategies; and a well-tuned sort environment.

IBM invests heavily in query optimization research and development. The DB2 II optimizer takes into account DB2 and DB2 II configuration options, standard

statistics from source data (such as cardinality or indexes), data server capability (such as join features or built-in functions), data server capacity, I/O capacity, and network speed. The following capabilities of the DB2 optimizer have a significant impact on the quality of the access plan generated:

- ▶ **Query rewrite** logic rewrites queries for more efficient processing. For example, it can convert a join of unions that drives a tremendous amount of data traffic, into a union of joins that leverages query power at the data server, and thereby minimizes data traffic back to the federated server. The database administrator (DBA) can define materialized query tables (MQTs), which the DB2 optimizer can transparently leverage via query rewrite to satisfy user queries.
- ▶ **Pushdown analysis** (PDA) capability identifies which operations can be executed at the data server prior to returning results to the federated server.

A number of DB2 and DB2 II configuration options are available to the DBA to impact the performance of a federated query on the federated server side. We will describe the impact of each of these options in turn, describe their pros and cons, provide best practices recommendations, and identify monitoring elements relevant to these options.

These options are organized, as follows, and as shown in Figure 3-6:

- ▶ Choice of Database Partition Feature (DPF)
- ▶ Federated query optimization
- ▶ Sort performance considerations
- ▶ Communication buffers RQRIOBLK

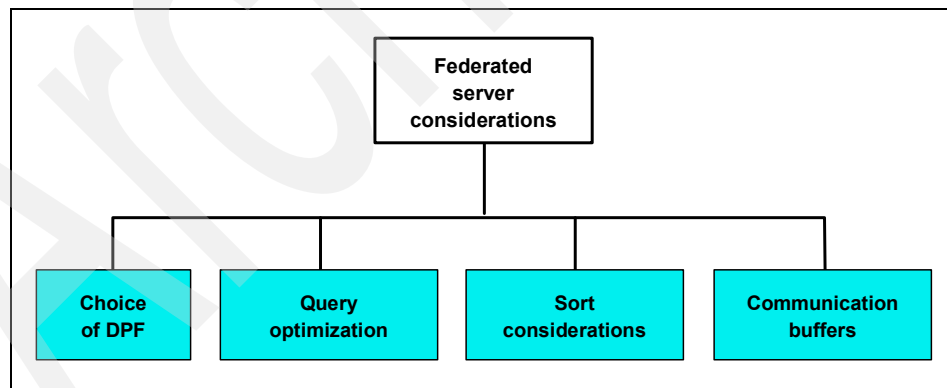


Figure 3-6 Federated server considerations topics

Choice of DPF

DB2 II may be installed in either a DPF or non-DPF environment. Typically a DB2 II DPF environment is deployed when federated queries need local access to local partitioned DB2 data as well as other remote data sources.

Attention: Do not consider partitioning a federated server database that contains no local partitioned data, or that contains local data that under normal circumstances would not be partitioned.

In a partitioned database environment, DB2 II supports exploitation of inter-partition parallelism for queries that reference local and remote data, or purely remote data via Computation Partition Groups (CPGs).

DPF with a computational partition node group defined may help performance of a standalone federated server if there are large intermediate result sets or significant SQL operations not pushed down. DPF gives the optimizer the option of distributing intermediate results to the nodes of a partitioned II instance to take advantage of parallel CPU and sortheap/bufferpool of multiple nodes.

To enable inter-partition parallelism for nickname data, the:

- ▶ DB2 II wrapper option `DB2_FENCED` *must be* set to “Y” (default is “N”).
- ▶ Computation process group (CPG) must be defined.
- ▶ There must be multiple nodes defined for the DB2 II instance.

DB2 II wrapper DB2_FENCED option

This option specifies whether the wrapper runs in trusted mode (`DB2_FENCED=N`) (which is the default) or fenced mode (`DB2_FENCED=Y`). Setting the fenced mode enables nickname data to be distributed to partitions for parallel processing with local partitioned data instead of joins occurring serially at the coordinator partition. Example B-11 on page 534 shows the exploitation of inter-partition parallelism for nickname data with fenced mode enabled.

The `DB2_FENCED` option may be changed from trusted to fenced and vice versa. To change the `DB2_FENCED` option to ‘Y’ for the wrapper named “drda”, issue the following statement:

```
ALTER WRAPPER drda OPTIONS (SET DB2_FENCED 'Y');
```

Performance considerations

Figure 3-7 on page 63 shows the wrapper architecture with the trusted and fenced mode.

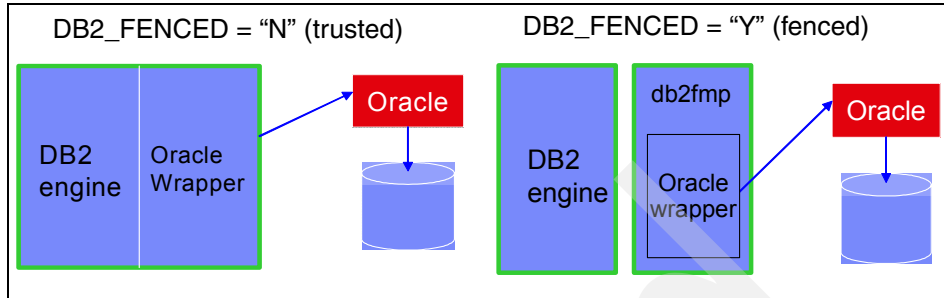


Figure 3-7 Wrapper architecture - Fenced and trusted

In trusted mode, all wrappers run in the same process as the DB2 engine. DB2 II loads the wrapper module into the db2agent process that was created when the user connected to DB2 II. The wrapper loads the data source client modules into the db2agent process in order to make the connection to the data source. While this is very efficient, unstable wrappers can bring down DB2. Another consideration is that this architecture does not allow resource sharing across applications and exploitation of parallelism, thereby increasing resource utilization and impacting scalability.

In fenced, each wrapper runs in isolated mode in a separate process. DB2 II creates another process, called a fenced mode procedure (fmp) process, and loads the wrapper module into this process. The wrapper loads the data source client modules into this fmp process so that the connection can be made to the data source. This isolation provides protection for the DB2 engine from errant wrappers, and enables resource sharing across applications and exploitation of parallelism. This in turn improves scalability via threading, as well as eases problem determination because of the ability to monitor individual processes. However, there is a cost associated with fenced wrappers because the communication between the engine and the fmp must take place via IPC resources.

Note: When a federated query accesses nicknames associated with different wrapper DB2_FENCED settings, the optimizer will consider inter-partition parallelism for nicknames associated with fenced mode, and serial processing for nicknames with trusted mode.

Best practices

We recommend the following:

- Use trusted mode for a wrapper:
 - When DB2 II is installed in a non-DPF environment.

- In DPF environments when nicknames are joined with low volumes of local DPF data. The DB2 optimizer is unlikely to choose inter-partition parallelism for nickname data in such cases. When nickname data distribution to the nodes does not occur, there is an avoidable overhead by eliminating the added cost of communication between the engine and the wrapper with trusted mode.
- ▶ Use fenced mode for a wrapper:
 - To enable parallelism in DPF environments.
 - To save memory in a high-concurrency environment.
 - To provide safety and fault isolation.
 - When large local partitioned tables are joined with nicknames.
 - In DPF environments for federated queries that only access and join large volumes of data from different data sources. This is to exploit the Computation Partition Group capability described in “Computation Partition Group (CPG)” on page 64.

To determine the setting of the DB2_FENCED wrapper option, issue the following SQL statement:

```
SELECT WRAPNAME, OPTION, SETTING FROM SYSIBM.SYSWRAPOPTIONS
WHERE WRAPNAME LIKE '%DRDA%'
```

Computation Partition Group (CPG)

A CPG defines a set of partitions for the optimizer to use for performing a dynamic redistribution operation for join operations. A CPG is a database partition group (other than IBMCATGROUP) that is specified in the system catalog SYSCAT.DBPARTITIONGROUPS.

Defining a CPG enables the optimizer to use an access plan that distributes nickname data to the partitions of the CPG. Defining a CPG enables inter-partition query parallelism for queries or parts of queries that reference only nicknames.

In a query plan that involves a CPG, the federated server redistributes nickname data across the partitions to create a parallel join. This type of plan can make queries run faster if the amount of nickname data that participates in the join is large.

The DB2_COMPPARTITIONGROUP registry variable specifies the CPG and can be set as follows:

```
db2set DB2_COMPPARTITIONGROUP=<partitiongroupname>
```

Where <partitiongroupname> is the name of the partition group that you want to define as the CPG.

Example 3-2 and Example 3-3 on page 66 show the access plan graph portion of **db2exfmt** output for a query with and without nickname exploitation of inter-partition parallelism as a result of CPG.

- ▶ The query shown in Example 3-1 is a join of two nicknames on a federated server installed in a DPF environment.
- ▶ Example 3-2 has the DB2_FENCED wrapper option set to 'N' (default), which is trusted mode. This access plan has one subsection (coordinator subsection) since all operations are serialized at the db2agent in a trusted mode environment.
- ▶ Example 3-3 on page 66 has the DB2_FENCED wrapper option set to 'Y', which is fenced mode.

This plan has four subsections, as follows:

- Subsection #1 reads the remote data coming from the SHIP operator 7 and redistributes the data (hashing) to the computational partition group through DTQ operator 6. This subsection is executed on the coordinator partition.
- Subsection #2 is similar to subsection #1 in that it reads the remote data coming from the SHIP operator 13 and redistributes the data to the computational partition group through DTQ operator 12 data (that is, hash partition the data to a particular partition based on the value of the join column). This subsection is executed on the coordinator partition.
- Subsection #3 reads the data coming from DTQ operators 6 and 12 and performs the merge scan join (MSJOIN operator 3). The subsection is executed on each node of the computational partition group.
- Coordinator subsection coordinates the other subsections. It distributes the subsections and then uses a table queue DTQ operator 2 to gather the results from the partitions in the computational partition group to be returned to the application.

Example 3-1 User query involving join of two nicknames

```
SELECT L_SHIPINSTRUCT FROM ORATPCD.LINEITEM 1 JOIN DB2TPCD.ORDERS x ON
L_ORDERKEY = O_ORDERKEY WHERE L_SHIPDATE BETWEEN DATE('1996-01-01') AND
DATE('1996-04-10')
```

Example 3-2 db2exfmt output showing no CPG exploitation

Access Plan:

Query Degree:0

```

      RETURN
      ( 1)
      Cost
      I/O
      |
      4629.63
      MSJOIN
      ( 2)
      23201.8
      48
      /---+---\
1111.11      4.16667
TBSCAN      FILTER
( 3)      ( 7)
1595.51      19795.6
48      0
|      |
1111.11      100000
SORT      TBSCAN
( 4)      ( 8)
1584.34      19795.6
48      0
|      |
1111.11      100000
SHIP      SORT
( 5)      ( 9)
1504.87      18795.6
48      0
|      |
10000      100000
NICKNM: ORATPCD      SHIP
LINEITEM      ( 10)
6885.6
0
|
100000
NICKNM: DB2TPCD
ORDERS
```

Example 3-3 db2exfmt output showing CPG exploitation

Access Plan:

Total Cost: 17785.5

Query Degree:1

Rows


```

RETURN
( 1)
Cost
I/O
|
4629.63
DTQ
( 2)
17785.5
48
|
2314.81
MSJOIN
( 3)
17676.9
48
/---+---\
555.555      4.16667
TBSCAN      FILTER
( 4)      ( 9)
1576.65      15194.9
48          0
|          |
555.555      50000
SORT      TBSCAN
( 5)      ( 10)
1571.03      15194.9
48          0
|          |
555.555      50000
DTQ      SORT
( 6)      ( 11)
1534.35      14694.9
48          0
|          |
1111.11      50000
SHIP      DTQ
( 7)      ( 12)
1504.87      9019.62
48          0
|          |
10000      100000
NICKNM: ORATPCD      SHIP
LINEITEM      ( 13)
6885.6
0
|
100000
NICKNM: DB2TPCD

```

Query optimization

As mentioned earlier, the selection of an optimal access plan through global optimization is critical to the performance of a federated query. The DB2 II optimizer takes into account DB2 and DB2 II configuration options, standard statistics from source data (such as cardinality or indexes), data server capability (such as join features or built-in functions), data server capacity, I/O capacity, and network speed.

The following options influence federated query optimization:

- ▶ Nickname statistics and index information
- ▶ Database manager configuration parameters INTRA_PARALLEL, MAX_QUERYDEGREE, SHEAPTHRES, SHEAPTHRES_SHR
- ▶ Database configuration parameters DFT_QUERYOPT, DFT_DEGREE, SORTHEAP, RQRIOLBK, buffer pool for temporary table space
- ▶ DB2 II wrapper option DB2_FENCED
- ▶ DB2 II server options COLLATING_SEQUENCE, COMM_RATE, CPU_RATIO, PUSHDOWN, DB2_MAXIMAL_PUSHDOWN, IO_RATIO, VARCHAR_NO_TRAILING_BLANKS
- ▶ DB2 II nickname options data type mappings, VARCHAR_NO_TRAILING_BLANKS, NUMERIC_STRING

We will review the influence of each of these options on federated query optimization according to the categories shown in Figure 3-8.

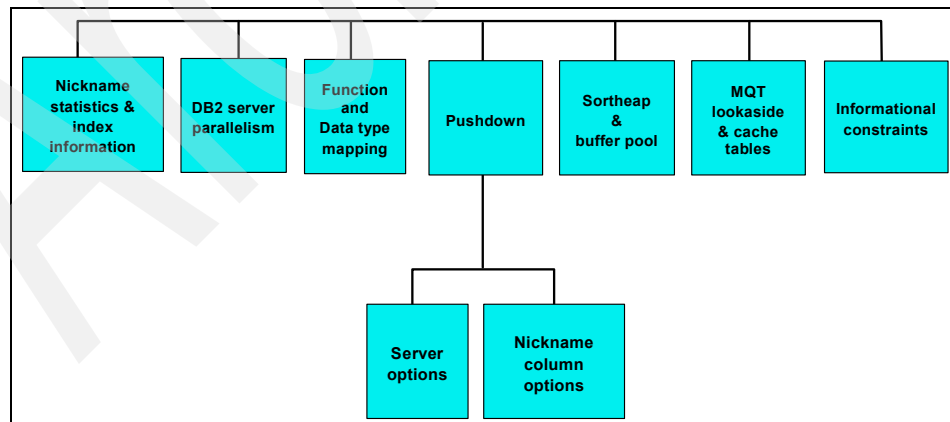


Figure 3-8 Query optimization topics

Nickname statistics and index information

For the DB2 optimizer to make superior access path decisions, it needs knowledge about available remote indexes on it, and accurate statistics about remote objects. The federated server relies on the remote source for its index and statistics information about each remote object. This information is retrieved when a nickname is created and stored in the federated server's global catalog.

Attention: The information in the global catalog is *not* automatically maintained if statistics on the remote object are refreshed or indexes are added or dropped.

It is the federated DBA's responsibility to ensure that the statistics and metadata information in the global catalog is kept in sync with the corresponding statistics and index information of the remote data objects.

If you believe that the federated server's information is out of sync with that of the remote data source, use the DB2 II provided stored procedure **NNSTAT** or the Statistics Update facility for synchronizing the remote objects' statistics with the statistics in the global catalog, as shown in Example 3-4. The Statistics Update facility can be invoked from the DB2 Control Center at regular intervals using the scheduler.

Example 3-4 NNSTAT stored procedure

```
CALL SYSPROC.NNSTAT( 'NULL',  
'ADMIN','STAFF','/home/iuser/reportlogs/log1.txt',?,?)
```

Example 3-4 retrieves currently available statistics on the STAFF nickname with a schema name of ADMIN, with the log records written to **log1.txt**.

The stored procedure **NNSTAT** or the Statistics Update facility performs the following steps for relational sources:

1. Create a “dummy” nickname on the remote data source.
 - If statistics were retrieved by the wrapper with this “dummy” nickname creation, update the global catalog with these collected statistics; if available, these statistics include CARD, FPAGES, OVERFLOW, COLCARD, HIGH2KEY, LOW2KEY, NLEAF, NLEVELS, CLUSTERFACTOR, CLUSTERRATIO, FULLKEYCARD, FIRSTKEYCARD.
 - If the local column name or type is different from the remote column name or type, then the update utility will *not* retrieve column statistics.

- If the local index name and definition is different from the remote index name or definition, the nickname statistics update facility will *not* retrieve index statistics.
- If statistics were *not* retrieved by the wrapper with this “dummy” nickname creation, then the **NNSTAT** stored procedure or the Statistics Update facility will execute minimum SQL to compute statistics (using logic similar to **get_stats** functionality) such as COUNT(*), COUNT(DISTINCT(colname)), MAX(colname), MIN(colname); this can be resource intensive.

Base statistics such as CARD, COLCARD, HIGH2KEY, LOW2KEY, FULLKEYCARD, and FIRSTKEYCARD are also collected.

2. Drop the “dummy” nickname after the global catalog has been updated.

While the **NNSTAT** stored procedure or the Statistics Update facility synchronizes the statistics of indexes that have the same column definitions at the remote data source and in the global catalog, it does *not* identify mismatches in remote and local index definitions. For example, assume that:

- ▶ The remote data source has indexes RIX1 (col1), RIX2 (col2) and RIX3 (col3) on table T1.
- ▶ Local indexes are NIX1 (col4), NIX2 (col2) and index specification NIX3 (col3) on the corresponding nickname NT1.

When the stored procedure is run, the statistics of NIX2 and NIX3 are synchronized. However, the **NNSTAT** stored procedure or the Statistics Update facility does *not* provide warnings or messages that:

- ▶ NIX1 index statistics are not updated and therefore may not be consistent with the other statistics.
- ▶ RIX1 has no corresponding index at the local server.

It is therefore up to the DBA to manually identify such mismatches and synchronize the index information and statistics as appropriate. This is discussed in “Indexes” on page 70 and “Statistics” on page 72.

▶ **Indexes**

When you create a nickname, DB2 II retrieves information about the indexes defined on the table at the remote source. This information is stored in the federated server’s global catalog as an attribute of the nickname, and is used during query optimization.

Index information for a nickname will *not* be retrieved if:

- The nickname for a table has no indexes.

- The nickname is for a remote view, Informix synonym, table structured file, Excel spreadsheet, or XML tagged file.

Views and Informix synonyms do not have index information in the data source catalog, but the tables referred to by the view or synonym may have indexes.

- The remote index has a column with more than 255 bytes, or a total index key length with more than 1024 bytes.
- The remote index is on LOB columns.

Another possible case in which index information for a nickname will be missing from the catalog is when you create a new index on a remote object *after* creating the nickname for the object. DB2 II is not notified of the change, and has no way of knowing that it needs to update its index information in the global catalog to include the new index.

To notify DB2 II of the existence of a missing index for a nickname, you can create an index specification to record information that includes the columns that comprise the index. However, this index specification will not include any statistical information by default. To gather statistical information on index specifications, you must ensure that both the index name and column names included in the index specification exactly match that of the remote index. Invocation of the Statistics Update facility or the NNSTAT stored procedure will then update the statistics of the index specification by reading information from the remote catalogs.

Similarly, when a nickname is created for a remote view, the federated server is unaware of the underlying tables (and their indexes) from which the view was generated. An index specification can be used to tell the DB2 optimizer about indexes on the underlying tables of a remote view, which may help it choose better access paths for queries involving the nickname to the remote view.

In either case, you supply the necessary index information to the global catalog using the CREATE INDEX... SPECIFICATION ONLY statement. No physical index is built on behalf of this nickname—only an entry is added to the system catalog to indicate to the query optimizer that such a remote index exists and whether it is a unique index. This helps the query optimizer in generating remote plans for relational nicknames. If you create a nickname for a view, and the view references a table that has a primary key, you can use 'CREATE UNIQUE INDEX...SPECIFICATION ONLY' to put an entry into the global catalog about this primary key. In the global catalog, the UNiquerule value for this entry will be 'U' rather than 'P', but 'U' is adequate to indicate to the DB2 II query optimizer the impact of the primary key for query optimization.

An index specification that defines a unique index also conveys the information about the uniqueness of the index columns to the federated system. Just like a regular unique index definition registered during relational nickname registration, such uniqueness information can help the query optimizer to generate a more optimal plan with strategies such as eliminating unnecessary DISTINCT operations.

► Statistics

DB2 stores statistical information on objects stored in the database including tables, table columns, and indexes. These statistics help the DB2 optimizer work out the best access plan for queries. In order to help the DB2 optimizer do its job, it is necessary to keep the statistics for each object in the database up to date. DB2 stores statistical information for nicknames as well. As nicknames are really just local references for remote tables, they look much like local tables to the DB2 optimizer. In fact, statistics for both local tables and nicknames are stored in the same way, and are accessible and updateable through DB2 system catalog views in the schema SYSSTAT.

DB2 stores the following types of nickname statistics:

- Table cardinality and page counts (SYSSTAT.TABLES) including CARD, FPAGES, NPAGES, and OVERFLOW
- Column cardinality (COLCARD) and column second lowest (LOW2KEY) and second highest (HIGH2KEY) values in SYSSTAT.COLUMNS
- Information on remote indexes for nicknames (SYSSTAT.INDEXES) including NLEAF, NLEVELS, CLUSTERFACTOR, CLUSTERRATIO, FULLKEYCARD, and FIRSTKEYCARD

The amount of statistical information stored for nicknames varies depending on the type of remote data source involved, as shown in Figure 3-9; for example, while table cardinality is available for nicknames on most sources, column second lowest and second highest values are only available for some sources.

Statistic:	MSSQL	Informix (IDS)	Oracle (Net8)	Sybase CTLIB	DRDA/ADB	Teradata	DRDA/MVS	DRDA/AS400
card		X	X	X	X	X	X	
npages		X	X	X	X	X	X	
fpages		X	X		X	X	X	
overflow			X		X			
colcard		X	X		X		X	
high2key		X			X			
low2key		X			X			
firstkeycard		X	X		X		X	
fullkeycard	X		X		X		X	
nlevels		X	X		X		X	
nleaf	X	X	X		X		X	
clusterratio		X	X		X		X	

Figure 3-9 Nickname statistics collected by data source

As mentioned earlier, nickname statistics and index information are retrieved from available information on the remote data source at the time that the nickname is created. Therefore, nickname statistics can only be as good as available remote statistics at nickname creation time. In particular, if no statistics have been collected on a remote object before a nickname is created, the nickname itself will not have any statistics. Similarly, if statistics are updated for an object on a remote data source, the new information is not automatically propagated to the corresponding DB2 nickname. Again, as discussed earlier, the same principle applies to indexes—DB2 is only aware of remote indexes for an object that is in existence at the time of nickname creation.

To make sure that DB2 nicknames have the best possible statistics and index data:

- Update statistics for objects on remote sources and create remote indexes *before* defining DB2 nicknames to them, so that DB2 can retrieve and store the current statistics information for the nickname.
- If updated statistics are collected for a remote object, or a new remote index is created, the DB2 statistics and index information for the corresponding nickname will be out of date. There is no **runstats** for nicknames. Use the **NNSTAT** stored procedure or the Statistics Update facility for updating statistics and manual procedures to synchronize index information.

Best practices

We recommend the following best practices for synchronizing the global catalog information with the corresponding remote object information:

1. When given a choice between creating a nickname for a table or for a view, always create the nickname for a table so that index information and statistics are automatically gathered when the nickname is created and when statistics are updated using the Statistics Update facility or the **NNSTAT** stored procedure. If you create a nickname for a view, you will have to obtain the statistics and index information for the tables in the view from the data source and update the nickname statistics and index information by command after the nickname is created.'

If you must create nicknames for views (not tables), obtain the statistics and index information for the tables referenced by the views.

Use CREATE INDEX...SPECIFICATION ONLY' and the SYSSTAT views to provide index information and statistics for the nicknames. An approach is to put your create nickname statements into scripts and follow each create nickname statement with the statements to create the index specifications and update the statistics.

2. Keep statistics up to date at data sources.

3. Run the Statistics Update facility or the **NNSTAT** stored procedure after statistics are updated at the data sources.
4. Ensure that the global catalog index and information for a nickname is always in sync with its referenced remote object. This requires judicious monitoring of changes occurring at the remote objects and then developing a schedule for synchronizing the two information sources. Use a combination of the **NNSTAT** stored procedure or the Statistics Update facility and the scheduler in the DB2 Control Center, and manual procedures to reestablish synchronization.
5. Consider adding index specifications for a nickname (for example, to indicate a specific sort order) even though they have no correspondence at the remote object, in order to provide the optimizer with additional information for generating a superior access plan.

Attention: Defining index specifications that violate the semantics of the remote object, such as defining a unique index on a column that contains duplicates, can return incorrect results. Extreme caution should be exercised when creating such index specifications.

The order of importance of index information and statistics for helping the optimizer pick good access plans is:

1. Index information—in particular which columns are indexed and whether the index is unique or there is a primary key. Use CREATE UNIQUE INDEX to put an entry into the global catalog about primary keys and unique indexes.
2. Cardinality of the view. CARD value for the nickname in SYSSTAT.TABLES.
3. Column distribution statistics for the nickname columns that will be used for joins, in filters, and in column functions.
 - Number of unique values—COLCARD in SYSSTAT.COLUMNS.
 - Maximum and minimum values—HIGH2KEY and LOW2KEY in SYSSTAT.COLUMNS. FULLKEYCARD and FIRSTKEYCARD in SYSSTAT.INDEXES are also important.

DB2 server parallelism

The influence of inter-partition parallelism and the DB2_FENCED wrapper option was covered in “Choice of DPF” on page 62, and the focus of this section is intra-partition parallelism.

The INTRA_PARALLEL, MAX_QUERYDEGREE, and DFT_DEGREE parameters determine whether intra-partition parallelism should be enabled, and the degree of parallelism desired. In a non-DPF environment, federated queries that involve local data can benefit from intra-partition parallelism. With federated

queries, the part of a query that involves local data can run in parallel, while the part that involves nicknames runs serially.

- ▶ The database manager configuration parameter `INTRA_PARALLEL` must be set to YES (default is NO) for the optimizer to consider intra-partition parallelism for federated queries.
- ▶ The database manager configuration parameter `MAX_QUERYDEGREE` sets an upper limit for the degree of parallelism for any query in the database. This value overrides the `CURRENT DEGREE` special register and the `DEGREE` bind option.
- ▶ The database configuration parameter `DFT_DEGREE` sets the default value for the `CURRENT DEGREE` special register and the `DEGREE` bind option.

Besides enabling `INTRA_PARALLEL`, set the `DFT_DEGREE` database configuration parameter to -1 or ANY, which sets the default value for the `CURRENT DEGREE` special register and the `DEGREE` bind option. If a query is compiled with `DEGREE = ANY`, the database manager chooses the degree of intra-partition parallelism based on a number of factors, including the number of processors and the characteristics of the query. The actual degree of parallelism used at run time may be lower than the number of processors, depending on these factors and the amount of activity on the system. Parallelism may be lowered before query execution if the system is heavily utilized in order to minimize adverse performance impact on other database users.

Another database configuration parameter that has an impact on query performance is `MAX_QUERYDEGREE`, which specifies the maximum degree of intra-partition parallelism that may be used for any SQL statement executing at the database manager level. This parameter should be set to the number of CPUs in the system to avoid the possibility of users inadvertently or intentionally setting their `CURRENT DEGREE` register value or `DEGREE` bind option too high.

The degree of parallelism chosen by the optimizer is displayed in `EXPLAIN` output, as shown by the Query Degree field in Example B-9 on page 490. The `Parallelism` field in the Database Context section in the `db2exfmt` output indicates whether intra-partition parallelism is enabled.

Performance considerations

While intra-partition parallelism can considerably improve response times of queries, it consumes additional CPU, I/O, real memory, and virtual memory (such as buffer pools, sort heaps, and other database heaps) resources and can exacerbate performance in a resource-constrained environment. Therefore, intra-partition should not be enabled in resource-constrained environments.

Best practices

We recommend the following best practices for tuning INTRA_PARALLEL, DFT_DEGREE and MAX_QUERYDEGREE: For SMP environments involving access to large volumes of local data, set INTRA_PARALLEL to YES, DFT_DEGREE to -1, and MAX_QUERYDEGREE to the number of CPUs, since parallelism could significantly enhance performance.

Performance monitoring metrics

To determine the degree of parallelism actually used at run time requires the use of the database system monitor.

Example 3-5 shows sort-relevant snapshot information from the **get snapshot for all applications** command with the DFT_MON_STMT switch set to ON.

Example 3-5 Database snapshot showing intra_parallelism monitor element

```
db2 => get snapshot for all applications
```

Application Snapshot

```
.....lines have been removed.....  
Degree of parallelism requested      = 1  
Number of agents working on statement = 1  
Number of subagents created for statement = 1  
.....lines have been removed.....
```

Important: All the fields in the snapshot monitor (whether they are water marks, counters, or gauges) should be monitored over many representative intervals spread over a few weeks, to detect consistent trends before reacting with configuration changes.

Note that counter type monitoring elements should be reset at the beginning of each monitoring interval by issuing the RESET MONITOR command.

The following fields are of interest for tuning the INTRA_PARALLEL, MAX_QUERYDEGREE, and DFT_DEGREE parameters:

- ▶ **Number of Agents Working on a Statement** is a gauge that describes the number of concurrent agents currently executing a statement or subsection. It is an indicator of how well the query is parallelized. This is useful for tracking the progress of query execution, by taking successive snapshots.
- ▶ **Number of Agents Created** is a high water mark that at the application level represents the maximum number of agents that were used when executing the statement. At the database level, it is the maximum number of agents for

all applications. It is an indicator of how well intra-query parallelism was realized.

- **Degree of Parallelism** is an information element that specifies the degree of parallelism requested when the query was bound. It is used with **Number of Agents Created** to determine if the query achieved the maximum level of parallelism.

Function and data type mapping

When nicknames reference non-DB2 relational sources such as Oracle or SQL Server, function and data type mapping may need to occur on the federated server for certain functions and columns of the remote object. DB2 II provides default mapping for most functions and data types, for example, the Oracle remote data type NUMBER is mapped to DOUBLE in DB2 II by default.

The default function and data type mappings are documented in the *IBM DB2 Information Integrator Federated Systems Guide Version 8.2*, SC18-7364-01.

The default type mappings are contained in the wrappers. The default type mappings can be over-ridden or extended by using the CREATE TYPE MAPPING statement, which puts entries into catalog view SYSCAT.TPEMAPPINGS. The CREATE TYPE MAPPING statement does not change the type mapping of columns of nicknames created before the CREATE TYPE MAPPING command was issued. Only nicknames created after the execution of the CREATE TYPE MAPPING statement are affected. To change the type mapping of an existing nickname column, use the following statement:

```
ALTER NICKNAME...ALTER COLUMN <colname> LOCAL TYPE <new type>
```

Performance considerations

When a federated query joins nickname columns related to different data sources, there is a likelihood that there will be a data type mismatch between the joined columns. In such cases the optimizer is unable to use the hash join as a join method, and performance may be impacted.

There are a couple of workarounds to use if the join columns at the different data sources cannot have the identical type, precision/length, and scale, as follows:

- Add an additional column to the table at one of the data sources, with the new column having the same type, precision/length, and scale as the join column at the other data source. Update this new column with values coming from the prior join column in the same table. Then create a unique index that includes the new column and update the statistics for the table. Then drop and re-create the nickname in DB2 II. Use the new nickname column in the join.
- Create a view over the table at one of the data sources. Cast the join column in the view to the same type, precision/length, and scale as the join column at

the other data source. Create a DB2 II nickname over the view. Keep the old nickname that was for the table referenced in the view; in fact, it is recommended that you drop and re-create that nickname or use the **NNSTAT** stored procedure or Statistics Update facility to update the statistics of the nickname. Then obtain the index information and statistics for the nickname over the table and use this information to update the statistics of the nickname for the view and to make CREATE INDEX...SPECIFICATION ONLY definitions for the nickname that is over the view. Use the nickname that is over the view in your join.

Best practices

We recommend that the default data type mappings of nickname columns that participate in joins with other data sources be reviewed for accuracy to ensure that the DB2 optimizer is not inhibited from choosing the most optimal access path available.

Pushdown

Pushdown is an important aspect of federated query processing. The “pushdown analysis (PDA) component of the SQL Compiler decides which parts of a query can be pushed down and processed remotely at the data sources.

The decision to push down certain parts and operators of a query depends on several factors, as follows:

- ▶ The availability of required functionality at the remote source
If the remote source is simply a file system with a flat file, then it is probably not possible to push down any filtering predicates.
- ▶ The options specified in the server definition of a remote source, as discussed in the DB2 server options and nickname column options bulleted list below
For instance, if the collating sequence at the federated server is different from the one at the remote server, then order dependent operations on string data and some predicates involved in the query have to occur at the federated server, and cannot be pushed down.

For further details, refer to *Using the federated database technology of IBM DB2 Information Integrator*, white paper by Anjali Grover, Eileen Lin, and Ioana Ursu, available from the Web site:

<http://www-3.ibm.com/software/data/pubs/papers/#iipapers>

As mentioned earlier, pushdown is influenced by the DB2 II server options and nickname column options, as follows:

- ▶ **DB2 II server options**

Server options may be set to persist over successive connections to the data source, and these values are stored in the global catalog. However, it is also possible to temporarily set a server option for the duration of a single connection to the database via the SET SERVER OPTION statement. This temporary setting is *not* stored in the global catalog. The advantage of this temporary setting is that it offers granular tuning control over a server option at a query level.

This section discusses the most common federated server options that can significantly impact the decisions made by the DB2 optimizer, and thereby on query performance:

- COMM_RATE, CPU_RATIO, IO_RATIO

These attributes describe the communication links to the remote source, and the relative speed of the remote system's CPU and I/O. By default, the federated server assumes that the remote machine is equal in power to the local machine, and that there is a 2 MB/sec link to it. Setting these options to indicate a more powerful remote machine or a faster link will tend to encourage query pushdown. These knobs are not perfect, but they are a way to indicate to the DB2 optimizer that a remote machine is fast or slow.

Tip: We recommend that you modify these parameters to accurately reflect the relative speeds, but use extreme caution or avoid modifying them altogether in order to influence the access plan.

- COLLATING_SEQUENCE

Setting this attribute to 'Y' tells PDA that the remote source sorts characters the same way that DB2 does. This means that the federated server can consider pushing down operations involving sorting, grouping, or inequality comparisons on CHAR and VARCHAR columns. For instance, setting COLLATING_SEQUENCE to 'Y' allows the DB2 optimizer to push down ORDER BY clauses that reference character and VARCHAR columns.

Pushdown of sorting, grouping, or inequality comparison operations on numeric, date, time, and date/time columns is *not* affected by this server option.

On a Microsoft SQL Server database server that is running Windows NT® or Windows 2000, the default collating sequence is case insensitive (for example, 'STEWART' and 'StewART' are considered equal). To guarantee correct results from the federated server, set the COLLATING_SEQUENCE server option to 'I'. This setting indicates that the Microsoft SQL Server data source is case insensitive. The federated

server does not push down queries if the results that are returned from the data sources will be different from the results that are returned when processing the query at the federated server. When you set the `COLLATING_SEQUENCE` server option to 'I', the federated server does not push down queries with string data or expressions and that include the following clauses, predicates, or functions:

- GROUP BY clauses
- DISTINCT clauses
- Basic predicates, such as equal to (=)
- Aggregate functions, such as MIN or MAX Related

Attention: We strongly recommend that this option be exercised with extreme caution, because you could get incorrect results if the remote source's collating sequence in reality did not match DB2's collating sequence after you set `COLLATING_SEQUENCE` to 'Y'.

Important: This setting can provide significant performance, and we recommend that the federated server be created in the same codepage and using the same collating sequence as the most frequently accessed data source. This will help avoid potentially expensive character translation and allow pushdown of order-dependent operations.

– VARCHAR_NO_TRAILING_BLANKS

This attribute is used for databases like Oracle that do not pad VARCHAR fields with trailing blanks. The SQL Compiler uses this information while checking any character comparison operations to decide the pushdown strategy to evaluate the operations.

DB2 uses blank padded comparison semantics while comparing character strings of unequal lengths. The comparison is made by using a copy of the shorter string that is padded on the right with blanks so that its length is equal to that of the longer string. This means that the string "A" is considered equivalent to "A " in DB2 UDB.

However, this behavior does not apply to all character data types across all data sources, such as VARCHAR and VARCHAR2 data types in Oracle.

In general, comparison operations on string columns without blank padding comparison semantics need to be evaluated locally unless the query compiler is able to find functions to enforce similar logic remotely. For example, the query compiler will add Oracle RTRIM function calls to varchar predicates, which are pushed down if `VARCHAR_NO_TRAILING_BLANKS = 'N'`. The RTRIM calls are

necessary in order to maintain data consistency, but will not allow the Oracle optimizer to use and index to access those RTRIM'ed varchar columns. This is likely to impact performance of both the query fragment pushed down to Oracle and the whole federated query.

If you are sure that your VARCHAR columns do not contain trailing blanks, then setting this option to 'Y' at the server level will allow the query compiler to push down query fragments without the RTRIM function, since it would be redundant because the data does not contain any trailing blanks.

The primary importance of the server/column option VARCHAR_NO_TRAILING_BLANKS on performance with Oracle data sources is:

- If VARCHAR_NO_TRAILING_BLANKS = 'N' (default), and an Oracle VARCHAR2 column is used in a join or filter, DB2 II will send 'RPAD(colname)= ' to Oracle to do the comparison or join and Oracle will not use an index to process the join or filter. Performance is often slow.
- If VARCHAR_NO_TRAILING_BLANKS='Y', and an Oracle VARCHAR2 column is used in a join or filter, DB2 II will send 'colname=' to Oracle to do the comparison or join and Oracle will use an index if it is available to process the join or filter. Performance is usually faster than if VARCHAR_NO_TRAILING_BLANKS='N'.

Attention: Here again, setting VARCHAR_NO_TRAILING_BLANKS to 'Y' when trailing blanks do exist at the remote data source can return inconsistent results.

Tip: This option can also be set at the nickname column level, which may be a better use of the option.

If it is set at the server level, the user would then need, for example, to ensure that *all* VARCHAR and VARCHAR2 columns of all data objects out of this Oracle data source are guaranteed not to contain trailing blanks.

– DB2_MAXIMAL_PUSHDOWN

This option specifies the primary criteria that the query optimizer uses when choosing an access plan. The query optimizer can choose access plans based on cost or based on the user requirement that as much query processing as possible be performed by the remote data sources.

With DB2_MAXIMAL_PUSHDOWN set to 'Y' (default is 'N'), reducing network traffic becomes the overriding criteria for the query optimizer. The query optimizer uses the access plan that performs the fewest number of SHIP operators in the plan regardless of cost. Setting this server option to 'Y' forces the federated server to use an access plan that might *not* be the lowest cost plan. Using an access plan other than the lowest cost plan can decrease performance. If a materialized query table (MQT) on the federated server can process part or all of the query, then an access plan that includes the materialized query table might be used. Using a materialized query table instead of pushing down operations to the data sources reduces network traffic.

Note: When the DB2_MAXIMAL_PUSHDOWN server option is set to 'Y', a query that will result in a Cartesian product will *not* push down the remote data sources. Queries that will result in a Cartesian product will be processed by the federated database.

The DB2_MAXIMAL_PUSHDOWN server option does not need to be set to 'Y' for the federated server to push down query processing to the remote data sources. When this server option is left to default ('N'), the query optimizer will push down query processing to the data sources. However, the primary criteria the optimizer uses when the option is set to 'N' is cost instead of network traffic.

Tip: Consider setting DB2_MAXIMAL_PUSHDOWN to 'Y' for poorly performing queries that have pushdownable predicates that are being executed at the federated server. Changing the default in this case should be the exception rather than the norm.

There are situations when DB2_MAXIMAL_PUSHDOWN of 'Y' will not help performance and may even hurt performance. We recommend adopting a trial and error method to determine whether setting DB2_MAXIMAL_PUSHDOWN to 'Y' will benefit or exacerbate a particular query's performance.

► **Nickname column options**

The NUMERIC_STRING and VARCHAR_NO_TRAILING_BLANKS and nickname column options impact the decisions made by the DB2 optimizer, and thereby query performance.

– **NUMERIC_STRING**

This nickname column option applies to character data types and is applicable to those data sources for which the COLLATING_SEQUENCE server option is set to 'N'.

The federated system does *not* push down any operations that can produce different results due to differences in collating sequences between the federated database and the remote data source. Suppose that a data source has a collating sequence that differs from the federated database collating sequence—in this case, the federated server typically does *not* push down sorts for any columns containing character data at the data source. It returns the retrieved data to the federated database, and performs the sort locally.

However, suppose that the column is a character data type (CHAR or VARCHAR) and contains only numeric characters (0 through 9)—this fact can be indicated to the DB2 optimizer by setting the NUMERIC_STRING column option to 'Y'. This gives the DB2 query optimizer the option of performing the sort at the data source because numbers are always sorted the same regardless of the collating sequence. If the sort is performed remotely, you can avoid the overhead of porting the data to the federated server and performing the sort locally.

Attention: Here again, setting NUMERIC_STRING to 'Y' should be used with extreme caution, since it can return inconsistent results when the column happens to contain non-numeric values at the remote data source.

– VARCHAR_NO_TRAILING_BLANKS

As discussed in “VARCHAR_NO_TRAILING_BLANKS” on page 80, this is also a server option. However, specifying this option at the column level provides greater flexibility and granularity if there are multiple tables in a database that do not all have missing trailing blanks.

Attention: Here again, setting VARCHAR_NO_TRAILING_BLANKS to 'Y' should be used with extreme caution, since it can return erroneous results when trailing blanks happen to exist for this column at the remote data source.

Sortheap and buffer pool

The SORTHEAP, SHEAPTHRES and SHEAPTHRES_SHR parameters impact query optimization and the efficiency of sorts. The size of the SORTHEAP and buffer pool for the temporary table space can impact the type of join (nested loop, merge scan or hash join) chosen. These parameters are discussed in more detail in “Sort considerations” on page 91.

Buffer pools are by far the component that can have the most dramatic impact on performance, since they have the potential to reduce application synchronous

I/Os. A buffer pool improves database system performance by allowing data to be accessed from memory instead of from disk. Because memory access is much faster than disk access, the less often the database manager needs to read from or write to a disk, the better the performance. A buffer pool is memory used to cache both user and system catalog table and index data pages as they are being read from disk or being modified. Federated queries that access local data would benefit from well-tuned buffer pools associated with local data.

A buffer pool is also used as overflow for sorts. In general, federated queries that do not access local data will perform most of their processing in the temporary table space in the federated server, and tuning the sort overflow buffer pool can improve federated query performance significantly.

IBMDEFAULTBP is the default buffer pool for the temporary tablespace. Sorts that overflow SORTHEAP spill to the temporary tablespace, and therefore the initial spill is into IBMDEFAULTBP. Therefore, it is recommended to increase the size of buffer pool IBMDEFAULTBP before SORTHEAP, since increasing the buffer pool will help to contain both sorts and transient temporary tables in memory. Increasing only SORTHEAP only benefits sorts and not temporary tables.

Note: DB2 II does not cache data from data sources into memory at the federated server for use in subsequent federated queries. DB2 II does not manage the updates to the data actually on the disk at the data source; therefore, if DB2 II were to keep data from data sources in memory and used that data in subsequent queries, the results could be wrong because the results would not reflect updates that were made to the data at the data source. Though DB2 II does use memory as needed for sorts and temp tables to process SQL operations not pushed down to data sources, these sorts and temp tables are transient, meaning that they and the data they held do not persist in memory or on disk at the federated server once the result of the query is given to the user. When you see the term *caching* used in conjunction with DB2 II, it refers to a different form a caching, which is the explicit creation and management of tables at the federated server with those tables containing data from data sources that is refreshed or replicated at user-specified intervals, as described in “MQT look-aside and cache tables” on page 85. The user, or the optimizer, can substitute these tables for nicknames to improve query performance, but the user needs to understand the latency of the data in these tables.

We recommend having a separate buffer pool for temporary tablespaces to improve sort performance, assuming adequate memory is available. Tuning buffer pools mainly involves improving the hit ratio. A discussion on tuning buffer pools is beyond the scope of this publication. Refer to the redbook *DB2 UDB*

ESE V8 non-DPF Performance Guide for High Performance OLTP and BI, SG24-6432; and the IBM DB2 UDB Administration Guide: Performance Version 8.2, SC09-4821, for a complete discussion of this topic.

MQT look-aside and cache tables

An MQT is a table whose structure and contents are based on an SQL query. The SQL query used in defining the MQT may access one or more tables or nicknames.

MQTs were designed to improve the performance of queries in a data warehousing environment where users often issue queries repetitively against large volumes of data with minor variations in a query's predicates. MQTs provide a look-aside capability for such queries that can result in orders of magnitude improvement in performance.

Since the MQT often contains precomputed summaries and/or a filtered subset of the data, it would tend to be much smaller in size than the base tables from which it was derived. When a user query accessing the base table is automatically rewritten (as shown in Figure 3-10 on page 86) by the DB2 optimizer to access the MQT instead, then significant performance gains can be achieved. MQTs do *not* require an aggregate function in its definition to be beneficial.

There are internal rules by which query rewrite substitutes MQTs for nicknames; when MQT substitution is done by query rewrite, the name of the MQT can be found in the Optimized Statement section in the **db2exfmt** output for the query. If query rewrite does not substitute an MQT for a nickname, the optimizer evaluates plans that use the nickname and plans that use the MQT. If the plan that uses the MQT is lower cost than the plan that uses the nickname, the optimizer will use the MQT. In that case, while the Optimized Statement section in the **db2exfmt** output still shows the nickname being used, the access plan graph section will show that it is actually the MQT that the optimizer selected to access.

Note: When the optimizer rewrites a query to reference an MQT, the access plan graph in the **db2exfmt** output identifies this action. However, in most cases, the Optimized Statement in the **db2exfmt** output shows the MQT only if it is an aggregate MQT.

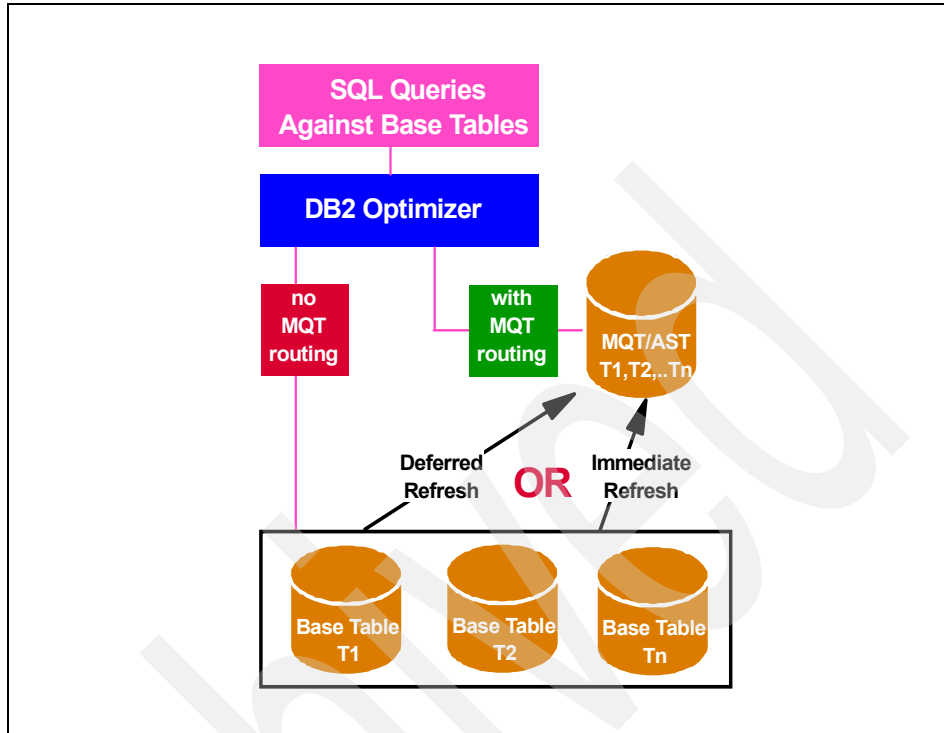


Figure 3-10 MQT/AST look-aside concept

Important: MQT functionality is somewhat similar to the role of a DB2 index, which provides an efficient access path that the query user is typically unaware of. However, unlike an index, a user may directly query the MQT, but this is not generally recommended since it would detract from the appeal of an MQT being a black box that an administrator creates and destroys as required to deliver superior query performance.

The refresh of an MQT may be immediate or deferred, as shown in Figure 3-10.

- ▶ With REFRESH IMMEDIATE MQTs, the contents of the MQT are automatically kept in sync at all times.
- ▶ With REFRESH DEFERRED, it is the DBA's responsibility to refresh the MQT at appropriate intervals—resulting in a latency that is dependent upon the refresh cycle chosen by the DBA.

MQTs that reference nicknames can only be defined as REFRESH DEFERRED.

Attention: We recommend creating MQTs on nicknames to minimize access to remote data sources when large volumes of data are involved. However, it should be noted that only queries that have the CURRENT REFRESH AGE special register set to ANY (the default is zero, which means that data must be current) will have their queries considered for query rewrite because of the REFRESH DEFERRED nature of MQTs defined on nicknames. This means that the DBA should consider creating such MQTs only when there is a significant query workload that tolerates latency explicitly by the setting of the CURRENT REFRESH AGE special register to ANY.

In DB2 II Version 8.2, there is a somewhat similar concept called cache tables, which provide a look-aside capability. A cache table can improve query performance by accessing a local subset of data instead of accessing data directly from the remote relational data source.

A cache table can be thought of as an enhanced MQT with the following properties:

- ▶ A nickname on the federated database system with the same column definitions as the remote relational data source table.
- ▶ One or more user-maintained MQTs that you defined on the nickname. The nickname can contain a subset of high-use data from a remote data source.
- ▶ A user-defined replication schedule that is associated with each cache table to automatically propagate changes from the data source object to the cache in an asynchronous manner.

There needs to be a row-to-row relationship between the data in the table at the data source and the MQTs created for the cache table. The cache table can have multiple MQTs, each with a different filter controlling which records from the source table are replicated into the MQT.

Figure 3-11 on page 88 illustrates the cache table concept.

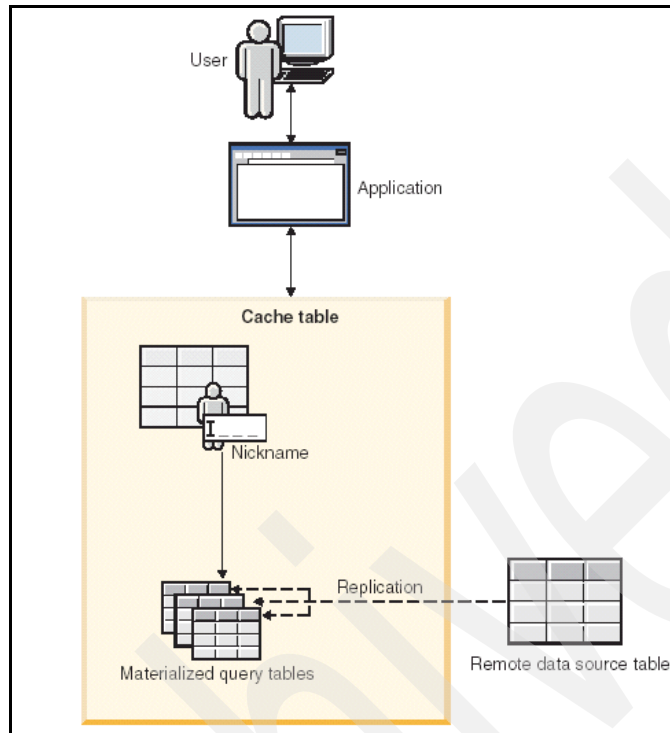


Figure 3-11 Cache table concept

Cache tables can be configured through the DB2 Control Center via a Cache Table Wizard. The wizard automatically creates an MQT over the nickname, configures and starts replication between the source and cache table, performs the initial refresh of the cache table, and sets the appropriate variables to enable query routing to cache tables. The cache table has the same name as the nickname component and contains all the columns of the remote object. A cache table can only be associated with one remote table. The cache table can be a full replica or partial subset of rows from your remote data source.

Applications that query a remote data source can query the cache table with no change to the application. During query processing, the optimizer directs the query to the cache table if it is able to satisfy the query, or the remote data source if the cache table cannot satisfy the query.

Cache tables provide the same benefits as look-aside as MQTs. However, they differ from the MQTs in that they use replication to propagate changes from the remote table to the cache table. The semantics of routing are slightly different in that the setting of the CURRENT REFRESH AGE special register is not considered by the optimizer when considering query rewrite to the cache table.

Instead, the routing to cache tables can be controlled by the setting of the default maintained table types for optimization (DFT_MTTB_TYPES) database configuration parameter, or the 'CURRENT MAINTAINED TABLE TYPES FOR OPTIMIZATION' special register. To enable cache table routing, 'FEDERATED_TOOL' must be listed in one of these parameters.

The following considerations apply to implementing cache tables:

- ▶ SQL replication is the mechanism used to maintain the cache table from the data source. Therefore, all capabilities and limitations of SQL replication apply to cache table maintenance. This includes the data sources supported, the requirement to have a primary key or unique index on the source table, and the mechanisms used for capturing changes at the data source. For example, an Oracle data source requires a trigger to be defined on the source table in order to capture changes; this can impact the performance of updates occurring at the data source.
- ▶ The cache table is set up via a wizard from the DB2 Control Center and includes the creation of the cache table and a primary key index on it, setting up the replication control tables and subscriptions, as well as the initial load of the cache table. However, it is up to the database administrator to define additional indexes on the cache table and collect statistics on the cache table and indexes via the DB2 **runstats** utility in order to provide the optimizer with up-to-date statistics for optimal access path selection.
- ▶ The initial refresh of the cache table is performed using SQL inserts only. For large volumes of data, the initial refresh may therefore be time consuming.

Note: Cache tables can only be set up via the DB2 Control Center. A cache table is a DB2 Control Center object and can be viewed as a folder in the navigation tree.

Attention: A technote is currently being written by DB2 II development that provides guidelines on the use of the cache table. You are strongly advised to review it when it becomes available, to ensure that cache tables are only used in appropriate scenarios.

Informational constraints

Informational constraints provide the DB2 optimizer with additional information to help it determine the optimal access path for a query. Informational constraints are implemented as check constraints or referential constraints, and are defined in the CREATE/ALTER TABLE statement with the NOT ENFORCED option.

Informational constraints are *not* enforced by the database manager during updates to a table; they are used by the DB2 optimizer for potential query rewrite.

Informational constraints can be used to improve the performance of queries with UNION ALL, as well as joins. It helps the optimizer rewrite outer joins as inner joins, and provides better estimates of cardinality. The DB2 optimizer can be directed to ignore an informational constraint by specifying the DISABLE QUERY OPTIMIZATION option in the CREATE/ALTER TABLE statement.

Important: When using informational constraints, ensure that an appropriate process is in place to ensure the integrity of the data.

Informational constraints are now supported for nicknames as well via the and CREATE/ALTER NICKNAME statement, and can therefore provide performance benefits by avoiding unnecessary join operations with remote data sources.

Examples of using informational constraints with nicknames include:

- ▶ Use with Cube Views

Informational constraints like primary key/foreign key between nicknames will allow the optimizer to use the MQTs created by Cube Views in queries where the nicknames are referenced. Without the informational constraints, the optimizer will use the nicknames even though Cube Views may have created MQTs for them.

- ▶ Check constraints

DB2 II will compare the predicate in a query with the check constraints on nicknames to determine if the nicknames should be used in the query. If the check constraint indicates that the nickname cannot contribute to the result, the nickname is not included in the access plan, thereby possibly improving performance.

- ▶ Primary key/foreign key

For queries that reference two nicknames (or a nickname and a table) with primary key/foreign key defined, and the query result does not require columns from one of the nicknames in the query. If the DB2 II optimizer determines that the RI constraint indicates that the nickname whose columns are not in the result will not add additional records to the result, then it excludes that nickname in the access plan, thereby possibly improving performance.

Attention: We urge extreme caution in the use of informational constraints that do not accurately reflect constraints that are in place on the remote table, since the user may get inconsistent results when the actual data happens to violate the integrity constraints asserted by the informational constraints.

Sort considerations

As mentioned earlier, SORTHEAP, SHEAPTHRES, SHEAPTHRES_SHR, and buffer pool impact query optimization and the efficiency of sorts. DB2 may perform sorts to return query results in a desired order (SELECT statement that uses the ORDER BY clause) when performing joins and during index creation.

Again, as mentioned earlier, federated queries that do not access local data will most likely perform most of their processing in the temporary table space in the federated server, and tuning the sort overflow buffer pool can improve federated query performance significantly. This section provides an overview of sort and the considerations in tuning sort-related parameters for superior performance.

The performance of sort depends upon many factors, including the writing of efficient SQL, and the configuring of the following parameters:

- ▶ Database manager configuration parameter
 - SHEAPTHRES is an instance-wide limit on the total amount of memory that can be consumed for sorts. It is used differently for private and shared sorts², as follows:
 - For private sorts, this parameter is an *instance-wide* soft limit on the total amount of memory that can be consumed by private sorts at any given time. When this limit is reached, the memory allocated for additional incoming private sort requests will be considerably reduced. These sorts are called *post threshold sorts*.
 - For shared sorts, this parameter is a *database-wide* hard limit on the total amount of memory that can be consumed by shared sorts at any given time. When this limit is reached, no further shared-sort memory requests are allowed (they will fail with SQL0955C) until the total shared-sort memory consumption falls below the limit specified by SHEAPTHRES.

Attention: This limit only applies to shared sorts when the SHEAPTHRES_SHR database configuration parameter is set to zero.

² DB2 performs shared sorts in database shared memory when the INTRA_PARALLEL database manager configuration parameter is enabled; otherwise it performs private sorts in private agent memory.

The default value is 20,000 4-K pages for UNIX and 64-bit platforms, and 10,000 4-K pages for Windows.

► Database configuration parameters

- SORTHEAP defines the maximum number of private memory pages to be used for a private sort, or the maximum number of shared memory pages to be used for a shared sort. Each sort operation has a separate sort heap that is allocated as needed by DB2 and freed when the sorting completes. In the case of a piped sort (see the definition in “Return of results from the sort phase” on page 93), the sort heap is not freed until the application closes the cursor associated with the sort. SORTs that are too large for SORTHEAP overflow to the temporary tablespace, which means they initially overflow into the buffer pool for the temporary tablespace. If that buffer pool is large enough and has free pages, the sort may still be contained in memory even though it overflowed SORTHEAP.

If directed by the DB2 optimizer, a smaller sort heap than the one specified by SORTHEAP is allocated by DB2.

The default is 256 4-K pages.

This parameter is configurable online.

- SHEAPTHRES_SHR is a *database-wide* hard limit on the total amount of database shared memory that can be used for shared sorts. When this limit is reached, no further shared-sort memory requests are allowed (they will fail with SQL0955C) until the total shared-sort memory consumption falls below the limit specified by SHEAPTHRES_SHR.

SHEAPTHRES_SHR is only meaningful when either the INTRA_PARALLEL database manager configuration parameter or the connection concentrator (database manager configuration parameters MAX_CONNECTIONS greater than MAX_COORDAGENTS) is enabled.

DB2 sorts are performed in two phases, as follows:

► Sort phase

When a sort is performed, DB2 allocates a block of memory equivalent to SORTHEAP in which data is sorted. When the sort cannot be sorted entirely within the sort heap, it overflows into the buffer pool and temporary table space, as shown in Figure 3-12 on page 93, and is called an *overflowed sort*. When no overflow occurs the entire sort is completed in the sort heap and is called a *non-overflowed sort*.

Attention: Sorts that do not overflow perform better than those that do.

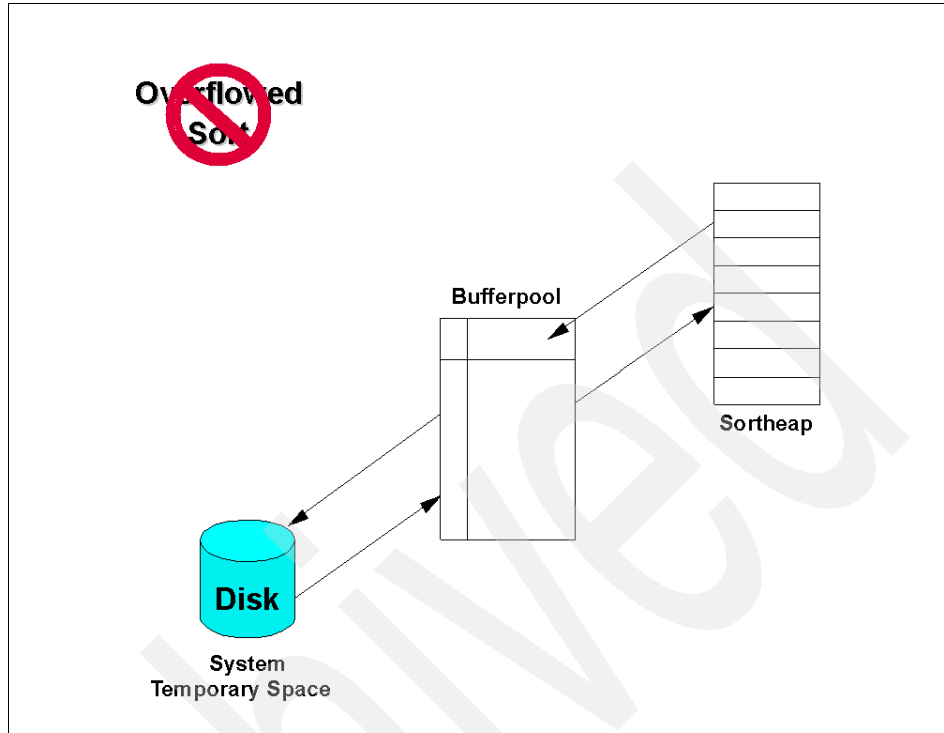


Figure 3-12 Overflowed sorts

► Return of results from the sort phase

If sorted information can return directly without requiring a temporary table to store the final sorted list of data, then it is called a *pipelined sort*. If the sorted information requires a temporary table to be returned, then it is called a *non-pipelined sort*. Figure 3-13 on page 94 shows an example of a non-overflowed pipelined sort.

Attention: A pipelined sort always performs better than a non-pipelined sort.

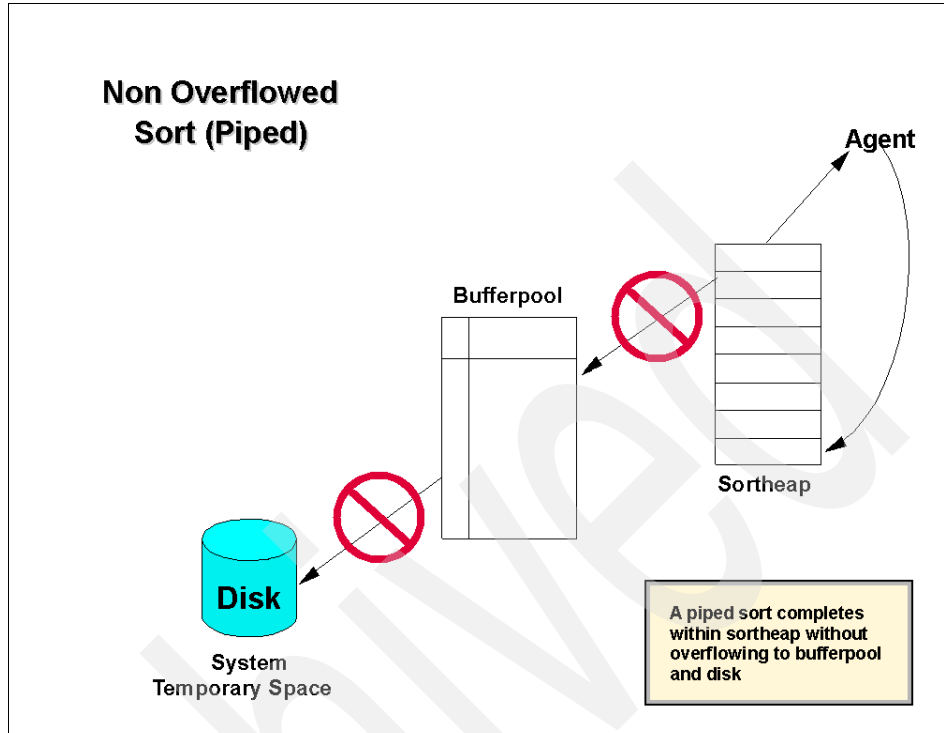


Figure 3-13 Non-overflowed piped sorts

Note: The DB2 optimizer will attempt to calculate the size of the sort heap that will be needed based on table statistics. If it requires more space than configured by SORTHEAP, then the sort will be overflowed; otherwise DB2 will attempt to allocate the entire SORTHEAP for the sort. In addition, the DB2 optimizer will also determine whether a piped or non-piped sort should be performed.

Performance considerations

Avoiding sorts is generally preferred through appropriate indexing of tables and writing of efficient SQL (Index SARGable predicates).

Note: However, in the case of federated queries with no access to local data, it is unlikely that indexing or writing of efficient SQL can avoid the occurrence of sorts.

If sorts cannot be avoided, then:

- ▶ Non-overflowed piped sorts are preferred by configuring an appropriate `SORTHEAP` value.
- ▶ Post threshold sorts should also be avoided by configuring an appropriate `SHEAPTHRES` value, since they result in smaller sort heap allocations and therefore have the potential for overflowed sorts.
- ▶ Avoid a rejection of new shared sorts (SQL0955C messages) due to an insufficient configuration of `SHEAPTHRES_SHR`.

Note: Over configuring these parameters without adequate real memory to back it up can result in system paging, which is detrimental to overall performance, even though non-overflow piped sorts are indicated.

With DB2 II, the determination whether to push down sorts for non-character nickname columns (that is, for `DATE`, `TIME`, `TIMESTAMP`, `INTEGER`, `SMALLINT`, `BIGINT`, `DECIMAL`, `FLOAT`, `DOUBLE`, `REAL`) is made by the optimizer based on cost.

For `CHAR` and `VARCHAR` nickname columns, pushdown analysis (PDA) first determines if the data source has the same collating sequence as the DB2 II database based on the setting of the `COLLATING_SEQUENCE` server option and `NUMERIC_STRING` column option. The default setting for both the `COLLATING_SEQUENCE` server option and the `NUMERIC_STRING` column option is 'N', unless the type of the data source is 'DB2/UDB' (that is, DB2 on Linux, UNIX, Windows).

- ▶ If the setting is 'N', then the PDA marks the sort as not-allowed-for-pushdown (`pushdown=0` in `db2trc`), and the optimizer is not allowed the option of pushing down the sort.
- ▶ If the setting is 'Y', then the sort is marked as allowed-for-pushdown (`pushdown=1` in `db2trc`), and the optimizer the option of deciding whether to push down the sort based on cost.

Best practices

We recommend the following best practices for enhancing sort performance and tuning the `SORTHEAP`, `SHEAPTHRES`, and `SHEAPTHRES_SHR` parameters:

- ▶ Avoid sorts as far as possible by defining appropriate indexes on tables and writing efficient SQL. Use the Design Advisor Wizard or `db2adv` command and `EXPLAIN` against all long-running queries to verify index access.
- ▶ Start with the default values, and tune `SORTHEAP`, `SHEAPTHRES`, and `SHEAPTHRES_SHR` to minimize overflows and non-piped sorts and then tune them for optimal values.

For critical workloads where frequent large sorts are performed, consider setting up a representative workload and tuning SORTHEAP, SHEAPTHRES, and SHEAPTHRES_SHR for that workload before adopting it in the production environment.

- ▶ Minimize sorts that cannot be pushed down.
 - For data sources on workstation platforms that have the same/similar code page to the DB2 II server, consider creating the DB2 II database with 'COLLATE USING IDENTITY' so that the DB2 II database and the workstation data sources will have the same collating sequence, and then set the server option COLLATING_SEQUENCE to 'Y'.

Restriction: It is not possible to make the collating sequence of the DB2 II database identical to the collating sequence of either the mainframe or the iSeries systems since the core codepages (EBCDIC on mainframe/iSeries versus ASCII on workstation) have differences.

- For data sources that do not have the same collating sequence as the DB2 II database, investigate the data that is in CHAR/VARCHAR columns that will be used in ORDER BY and WHERE < /WHERE > clauses. If all numeric characters or if a sort at the data source will yield the same result as a sort for the same data at the DB2 II server, then set column option NUMERIC_STRING to 'Y'.
- ▶ For sorts that cannot be pushed down, estimate their size using DB2 II explain tools and increase either SORTHEAP or IBMDEFAULTBP so that the sorts will be contained in memory.

Performance-monitoring metrics

Metrics for monitoring SORTHEAP, SHEAPTHRES, and SHEAPTHRES_SHR may be obtained from the snapshot monitor and appropriate sorting health indicators in the Health Center.

Figure 3-14 on page 97 shows sort-relevant snapshot information from the **get snapshot for dbm** command, and Figure 3-15 on page 97 shows sort-relevant information from the **get snapshot for db** command.

```

db2 get dbm snapshot

Private Sort heap allocated           = 0
Private Sort heap high water mark    = 4536
Post threshold sorts                  = 0
Piped sorts requested                 = 143
Piped sorts accepted                  = 143

```

Figure 3-14 Database manager snapshot showing sort monitor elements

```

db2 get snapshot for database on DTW

Database Snapshot

Database name                         = DTW
Total Private Sort heap allocated     = 0
Total Shared Sort heap allocated      = 0
Shared Sort heap high water mark      = 0
Total sorts                           = 18
Total sort time (ms)                  = 18086
Sort overflows                        = 8
Active sorts                          = 0

```

Figure 3-15 Database snapshot showing sort monitor elements

Important: All the fields in the snapshot monitor (whether they are water marks, counters, or gauges) should be monitored over many representative intervals spread out over a few weeks, to detect consistent trends before reacting with configuration changes.

Note that counter type monitoring elements should be reset at the beginning of each monitoring interval by issuing the RESET MONITOR command.

The following fields are of interest for tuning SORTHEAP, SHEAPTHRES, and SHEAPTHRES_SHR:

- ▶ Database manager snapshot
 - **Private sort heap high water mark** is a water mark of the total number of allocated pages of sort heap space for all private sorts across all databases.

- **Post threshold sorts** is a counter that records the total number of sorts that have requested heaps after the sort heap threshold (SHEAPTHRES) has been exceeded.
- **Piped sorts requested** is a counter that records the total number of piped sorts that have been requested.
- **Piped sorts accepted** is a counter that records the total number of piped sorts that have been accepted. A piped sort is rejected if the sort heap threshold (SHEAPTHRES) will be exceeded when the sort heap is allocated for the sort.
- ▶ Database snapshot
 - **Total Private Sort heap allocated** is a gauge that records the total number of allocated pages of sort heap space for all private sorts.
 - **Total Shared Sort heap allocated** is a gauge that records the total number of allocated pages of sort heap space for all shared sorts.
 - **Shared Sort heap high water mark** is a water mark of the total number of allocated pages of sort heap space for all shared sorts for this database.
 - **Total sorts** is a counter that records the total number of sorts that have been executed.
 - **Total sort time (ms)** is a counter that records the total elapsed time in milliseconds for all sorts that have been executed.
 - **Sort overflows** is a counter that records the total number of sorts that ran out of sort heap and may have required disk space for temporary storage.
 - **Active sorts** is a gauge that records the number of sorts in the database that currently have a sort heap allocated.

Note: Database manager and database snapshots only indicate the presence of a sort problem, not the SQL statements causing them. In order to identify the SQL statements invoking sorts, a dynamic SQL snapshot or an event monitor for SQL statements is required.

Consider modifying SORTHEAP, SHEAPTHRES, and SHEAPTHRES_SHR under the following circumstances:

- ▶ Compare the water marks for **Private Sort heap high water mark** and **Total Shared Sort heap high water mark** with the corresponding values of SHEAPTHRES and SHEAPTHRES_SHR.

If they are significantly lower than the configuration parameters, consider setting the configuration parameters to a few percentage points above the water mark levels.

If **Private Sort heap high water mark** is higher than SHEAPTHRES, then it means that less than the desired amount of sort heap is being allocated. This should register as **Post threshold sorts**. Increase the SHEAPTHRES value a few percentage points above the water mark.

If **Shared Sort heap high water mark** is close or equal to the SHEAPTHRES_SHR configuration parameter setting, increase its value a few percentage points above the water mark. User applications may receive the SQL0955C error message.

- ▶ If the percentage of overflowed sorts (Sort overflows / Total sorts) is high, increase SORTHEAP and/or SHEAPTHRES.
- ▶ If **Post threshold sorts** relative to **Total sorts** are high, increase SHEAPTHRES and/or decrease SORTHEAP.

Decreasing SORTHEAP may help since smaller allocations are less likely to cause the SHEAPTHRES threshold to be exceeded. However, decreasing SORTHEAP may cause sort overflows to occur, which is undesirable.

- ▶ If piped sorts rejected (**Piped sorts requested - Piped sorts accepted**) relative to **Piped sorts requested** is high, increase SORTHEAP or SHEAPTHRES.
- ▶ If there are user complaints about receiving SQL0955C error messages (which indicate that the shared memory sorts threshold has been exceeded), increase SHEAPTHRES_SHR.
- ▶ Federated environments will traditionally have a number of small and large sorts, and the settings of SORTHEAP, SHEAPTHRES, and SHEAPTHRES_SHR parameters as well as appropriate buffer pool assignments and proper temporary table space placement will have a significant impact on sort performance.

Communication buffers

This RQRIOLBK database manager configuration parameter specifies the size of the communication buffer between *remote* applications and their database agents on the database server. When a database client requests a connection to a remote database, this communication buffer is allocated on the client. On the database server, a communication buffer of 32767 bytes is initially allocated, until a connection is established and the server can determine the value of RQRIOLBK at the client. RQRIOLBK also determines the maximum size of blocks that DB2 II can use in communication with the data source client software.

Note: In a federated server environment accessing remote data sources, the client is the federated server, and the remote data source is the *database* server.

Once the database server knows this value, it will reallocate its communication buffer if the client's buffer is not 32767 bytes. In addition to this communication buffer, this parameter is also used to determine the I/O block size at the database client when a blocking cursor is opened. This memory for blocked cursors is allocated out of the application's private address space, so you should determine the optimal amount of private memory to allocate for each application program. If the database client cannot allocate space for a blocking cursor out of an application's private memory, a non-blocking cursor will be opened.

The communication buffer is allocated when:

- ▶ A remote client application issues a connection request for a server database.
- ▶ A blocking cursor is opened, additional blocks are opened at the client.

The communication buffer is deallocated when:

- ▶ The remote client application disconnects from the server database.
- ▶ The blocking cursor is closed.

The default value is 32767 bytes and the maximum is 65535 bytes, and this parameter is *not* configurable online.

Performance considerations

If the request to the database manager, or its associated reply, does not fit into the buffer, it will be split into two or more send-and-receive pairs. The size of this buffer should be set to handle the majority of requests using a single send-and-receive pair.

You should also consider the effect of this parameter on the number and potential size of blocking cursors. Large row blocks may yield better performance if the number or size of rows being transferred is large; for example, if the amount of data is greater than 4096 bytes. However, the trade-off is that the larger record blocks increase the size of the working set memory for each connection.

Note: The size of the request is based on the storage required to hold the input SQLDA, all of the associated data in the SQLVARs, the output SQLDA, and other fields that do not generally exceed 250 bytes.

Best practices

We recommend the following best practices for tuning RQRIOBLK:

1. If your application's requests are generally small and the application is running on a memory-constrained system, you may wish to reduce the value of this parameter. If your queries are generally very large, requiring more than one send and receive request, and your system is not constrained by memory, you may wish to increase the value of this parameter.

2. Compute RQRIOBLK as follows if the information is available:

$$\text{rqrioblk} = ((\text{size of input SQLDA}) + (\text{size of each input SQLVAR}) + (\text{size of output SQLDA}) + 250))$$
3. Let RQRIOBLK default to 32767 bytes and then tune for optimal value.
4. If there will be large results from the data source to II:
 - a. Increase network speed between the DB2 II server and the data source.
 - b. Decrease the number of routers between the DB2 II server and the data source.
 - c. Increase the network packet size.
 - d. Examine/increase the communication buffer or packet size between the data source client and the data source server, for example, Sybase packet_size.
 - e. Increase RQRIOBLK.

Performance-monitoring metrics

DB2 does not provide a direct mechanism to monitor and tune RQRIOBLK. You instead need to monitor other elements in an application snapshot that indirectly relates to RQRIOBLK. The monitoring elements of interest in an application snapshot, as generated by a **get snapshot for applications** command, are shown in Example 3-6.

Example 3-6 Snapshot showing block remote cursor information

Rejected Block Remote Cursor requests	= 50
Accepted Block Remote Cursor requests	= 1000

Important: All the fields in the snapshot monitor (whether they are water marks, counters, or gauges) should be monitored over many representative intervals spread over a few weeks, to detect consistent trends before reacting with configuration changes.

Note that counter type monitoring elements should be reset at the beginning of each monitoring interval by issuing the RESET MONITOR command.

The fields of interest in Example 3-6 for tuning RQRIOBLK are:

- **Rejected Block Remote Cursor requests** is a counter that records the number of times a request for a blocking cursor at the database server was rejected and the request was converted to a non-blocking cursor.

If there are many cursors blocking data, the communication heap may become full. When this heap is full, an error is *not* returned; instead, no more

I/O blocks are allocated for blocking cursors. If cursors are unable to block data, performance can be affected.

- **Accepted Block Remote Cursor requests** is a counter that records the number of times a request for an I/O block at the database server was accepted.

Compute the following metric for tuning purposes. The percentage of rejected block remote requests (PRBRR) is as follows:

$$\text{PRBRR} = ((\text{Rejected Block Remote Cursor requests}) / (\text{Accepted Block Remote Cursor requests} + \text{Rejected Block Remote Cursor requests})) * 100$$

Consider increasing RQRIOLBK:

- If PRBRR is consistently high; QUERY_HEAP_SZ should also be increased correspondingly.
- When applications receive SQL1221N or SQL1222N messages or see DIA3605C in the **db2diag.log**, increase RQRIOLBK incrementally until these conditions do not reappear.

The number of send and receive requests occurring for a particular application can be found by enabling the CLI trace facility.

3.4.3 Data source considerations

The DB2 optimizer generates an access plan based on a number of factors, as discussed in “Query optimization” on page 68, and then generates a query fragment to be executed at the remote data source.

This query fragment sent to the data source for execution can be identified in the RMTQTXT field of the SHIP operator in **db2exfmt** output for the query, as shown in Example 3-7.

If it is suspected that the query fragment sent to the remote data source is not performing well, then it can be tuned according to the unique considerations associated with the particular data source by that data source administrator.

Example 3-7 db2exfmt output with query fragment in RMTQTXT of SHIP operator

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

***** EXPLAIN INSTANCE *****

DB2_VERSION: 08.02.0
SOURCE_NAME: TOOL1E00
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-07-02-19.24.26.071589
EXPLAIN_REQUESTER: KAWA

Database Context:

Parallelism: None
CPU Speed: 4.723442e-07
Comm Speed: 100
Buffer Pool size: 78000
Sort Heap size: 20000
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 1020

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 1 -----

QUERYNO: 1
QUERYTAG:
Statement Type: Select
Updatable: No
Deletable: No
Query Degree: 1

Original Statement:

**SELECT O_ORDERPRIORITY, COUNT(*) AS ORDER_COUNT
FROM ORA.ORDERS
WHERE O_ORDERKEY BETWEEN 1 AND 1000000 AND EXISTS
(SELECT *
FROM DB2.LINEITEM
WHERE L_ORDERKEY = O_ORDERKEY AND L_COMMITDATE < L_RECEIPTDATE)
GROUP BY O_ORDERPRIORITY
ORDER BY O_ORDERPRIORITY**

Optimized Statement:

```
-----  
SELECT Q4.$C0 AS "O_ORDERPRIORITY", Q4.$C1 AS "ORDER_COUNT"  
FROM  
  (SELECT Q3.$C0, COUNT(*) )  
  FROM  
    (SELECT DISTINCT Q2.O_ORDERPRIORITY, Q2.$P-ROWID$  
      FROM DB2.LINEITEM AS Q1, ORA.ORDERS AS Q2  
      WHERE (Q1.L_ORDERKEY = Q2.O_ORDERKEY) AND (Q1.L_COMMITDATE <  
        Q1.L_RECEIPTDATE) AND (Q2.O_ORDERKEY <= 1000000) AND (1 <=  
        Q2.O_ORDERKEY)) AS Q3  
  GROUP BY Q3.$C0) AS Q4  
ORDER BY Q4.$C0
```

Access Plan:

```
-----  
Total Cost: 222926  
Query Degree:1
```

```
Rows  
RETURN  
( 1)  
Cost  
I/O  
|  
5  
GRPBY  
( 2)  
222926  
16810.7  
|  
250002  
TBSCAN  
( 3)  
222896  
16810.7  
|  
250002  
SORT  
( 4)  
222762  
16810.7  
|  
250002  
MSJOIN  
( 5)  
222162  
16810.7  
/---+---\
```

250002	1.99876
SHIP	FILTER
(6)	(9)
37714.3	184277
9486.34	7324.35
1.5e+07	499694
NICKNM: ORA	SHIP
ORDERS	(10)
	184277
	7324.35
	5.99861e+07
	NICKNM: DB2
	LINEITEM

1) RETURN: (Return Result)

Cumulative Total Cost: 222926
 Cumulative CPU Cost: 4.89221e+09
 Cumulative I/O Cost: 16810.7
 Cumulative Re-Total Cost: 222327
 Cumulative Re-CPU Cost: 3.62365e+09
 Cumulative Re-I/O Cost: 0
 Cumulative First Row Cost: 222794
 Estimated Bufferpool Buffers: 0
 Remote communication cost:397356

Arguments:

 BLDLEVEL: (Build level)
 DB2 v8.1.1.64 : s040509
 ENVVAR : (Environment Variable)
 DB2_EXTENDED_OPTIMIZATION = ON
 STMTHEAP: (Statement heap size)
 8192

Input Streams:

9) From Operator #2

Estimated number of rows: 5
 Number of columns: 2
 Subquery predicate ID: Not Applicable

Column Names:

+Q5.O_ORDERPRIORITY(A)+Q5.ORDER_COUNT

.....lines have been removed.....

6) SHIP : (Ship)

Cumulative Total Cost: 37714.3
Cumulative CPU Cost: 4.40279e+08
Cumulative I/O Cost: 9486.34
Cumulative Re-Total Cost: 168.177
Cumulative Re-CPU Cost: 3.56048e+08
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 100.032
Estimated Bufferpool Buffers: 9486.51
Remote communication cost:139105

Arguments:

CSERQY : (Remote common subexpression)
FALSE

DSTSEVER: (Destination (ship to) server)
- (NULL).

JN INPUT: (Join input leg)
OUTER

RMTQTX : (Remote statement)

**SELECT A0."O_ORDERKEY", A0."O_ORDERPRIORITY", A0.ROWID FROM
"IITEST"."ORDERS" A0 WHERE (1 <= A0."O_ORDERKEY") AND (A0."O_ORDERKEY" <=**
1000000) ORDER BY 1 ASC

SRCSEVER: (Source (ship from) server)
ORASERV

STREAM : (Remote stream)
FALSE

Input Streams:

1) From Object ORA.ORDERS

Estimated number of rows: 1.5e+07
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$

Output Streams:

2) To Operator #5

Estimated number of rows: 250002
Number of columns: 4
Subquery predicate ID: Not Applicable

Column Names:

+Q2.0_ORDERKEY(A)+Q2.\$RID\$+Q2.\$P-ROWID\$
+Q2.0_ORDERPRIORITY

.....lines have been removed.....

10) SHIP : (Ship)

Cumulative Total Cost: 184277
Cumulative CPU Cost: 2.47251e+09
Cumulative I/O Cost: 7324.35
Cumulative Re-Total Cost: 253.012
Cumulative Re-CPU Cost: 5.35653e+08
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 65.049
Estimated Bufferpool Buffers: 7325.35
Remote communication cost:258251

Arguments:

CSERQY : (Remote common subexpression)
FALSE
DSTSEVER: (Destination (ship to) server)
- (NULL).

RMTQTX : (Remote statement)

SELECT AO."L_ORDERKEY" FROM "TPCD"."LINEITEM" AO WHERE
(AO."L_COMMITDATE" < AO."L_RECEIPTDATE") AND (AO."L_ORDERKEY" <= 1000000) AND
(1 <= AO."L_ORDERKEY") ORDER BY 1 ASC, AO."L_COMMITDATE" ASC,
AO."L_RECEIPTDATE" ASC FOR READ ONLY

SRCSEVER: (Source (ship from) server)
DB2SERV
STREAM : (Remote stream)
FALSE

Input Streams:

3) From Object DB2.LINEITEM

Estimated number of rows: 5.99861e+07
Number of columns: 4
Subquery predicate ID: Not Applicable

Column Names:

+Q1.\$RID\$+Q1.L_RECEIPTDATE+Q1.L_COMMITDATE
+Q1.L_ORDERKEY

Output Streams:

4) To Operator #9

Estimated number of rows: 499694
Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:

+Q1.L_ORDERKEY(A)+Q1.L_COMMITDATE(A)
+Q1.L_RECEIPTDATE(A)

Objects Used in Access Plan:

Schema: DB2

Name: LINEITEM

Type: Nickname

Time of creation: 2004-06-11-21.33.19.482113
Last statistics update: 2004-06-11-22.38.00.532171
Number of columns: 16
Number of rows: 59986052
Width of rows: 133
Number of buffer pool pages: 2081039
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: ORA

Name: ORDERS

Type: Nickname

Time of creation: 2004-06-11-21.33.10.349783
Last statistics update: 2004-06-11-22.32.45.545670
Number of columns: 9
Number of rows: 15000000
Width of rows: 49
Number of buffer pool pages: 443840

Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Best practices

We recommend the following best practices for tuning data sources:

- ▶ If there is a choice between creating nicknames for tables or for views, create nicknames for tables.
- ▶ Keep statistics of tables at data sources current.
- ▶ Tune the data source for SQL statements coming from DB2 II, such as adding indexes.

3.4.4 Efficient SQL queries

In general, all the best practices associated with writing efficient SQL for non-federated queries apply to federated environments as well. Refer to the redbook *DB2 UDB ESE V8 non-DPF Performance Guide for High Performance OLTP and BI*, SG24-6432; and the *IBM DB2 UDB Administration Guide: Performance Version 8.2*, SC09-4821, for a complete discussion of this topic.

The redbook *DB2 UDB's High-Function Business Intelligence in e-business*, SG24-6546, provides guidelines on SQL coding practices to encourage MQT routing.

The following additional considerations apply to creating nicknames and federated queries. Best practices can be broadly classified as being nickname related, query related, or miscellaneous.

Nickname-related best practices

In general, nickname-related best practices apply to DBAs as follows:

- ▶ Ensure that nickname index definitions and statistics are current with respect to the remote data source. This is by far the most significant factor that can positively influence the performance of a federated query, since it would facilitate the generation of a superior access plan.
- ▶ Define informational referential integrity and functional dependency constraints on nicknames if appropriate. This provides the DB2 optimizer with additional information that allows it to consider rewriting the query into a more efficient form.

Attention: If the data actually violates informational constraints that have been defined, inconsistent results may be returned.

- ▶ When multiple tables at a single remote data source need to be joined in a query, consider creating a view at the remote data source of such a join and a nickname on this view. This facilitates simpler user queries as well as pushdown in most cases.

The disadvantage of creating nicknames over remote views is that it hides the details of the query from the optimizer and requires the DBA to manually update the statistics in the global catalog since statistics are not captured automatically on nickname creation on a view.

Attention: This approach is not generally recommended except in extenuating circumstances such as pushdown not occurring with nicknames on remote tables.

- ▶ When defining the local data types for a nickname column that is joined with columns of other nicknames/tables, ensure that the local data types of the columns involved match perfectly in terms of scale and precision. Mismatched data types may result in less pushdown, or the exclusion from consideration of merge scan and hash joins for access path selection.

This may require modifying the default data type mapping provided by DB2 II for a given data source.

- ▶ If there is a mismatch in data type, precision/length, and scale between join columns of two nicknames, the techniques to make them alike are:
 - Alter the nickname to change the local type of the columns of one of the nicknames to match the type, precision/length, and scale of the join column of the other nicknames. This is only allowed if the new data type is compatible with the existing data type. Also, this should not be done if it will cause value truncation or padding that will cause join results to be invalid.
 - Add a column to the table at one of the data sources with the new column having the same type, length/precision, and scale as the join column in the other data source. Update the new column with values from the former join column in the same table. Create a unique index that includes the column and update statistics for the table. Then drop and re-create the nickname for the table and use the new column in the join.
 - Create a view at one of the data sources in which the join column is cast with the same type, length/precision, and scale as the join column in the other data source. Create a new nickname for the view, but keep the old

nickname for the table. Obtain the index information and statistics from the DB2 II catalog for the old nickname on the table. Use this information to CREATE INDEX...SPECIFICATION only, and update the statistics for the new nickname that is for the view. Use the new nickname in the join. When you want to update the statistics for the nickname for the view, drop/re-create or run the Statistics Update facility or the **NNSTAT** stored procedure for the old nickname for the table, and then use the index information and statistics for this old nickname to update the index information and statistics for the nickname that is for the view.

Query-related best practices

In general, query-related best practices apply to the application programmer, as follows:

- ▶ Consider using the SET SERVER OPTION to temporarily change the server options for a given query to override default or global settings to fine tune it. This will minimize the performance impact on other queries that benefit from the server settings recorded in the global catalog.
- ▶ Organize queries involving outer joins more efficiently by using parentheses to group the same server nickname outer join together, as shown in Example 3-8.

Note: In DB2 V8.2 when DB2_MAXIMAL_PUSHDOWN is set to 'Y', the outer joins can be reordered and grouped together without users having to do this themselves.

Example 3-8 Writing outer joins more efficiently

```
--Assume slt1 and slt2 are two nicknames on server s1
--Assume s2t1 and s2t2 are two nicknames on server s2
-----
-----
-- Poorly written outer join query
select *
from slt1    left outer join s2t1 on slt1.c1 = s2t1.c1
             left outer join s2t2 on s2t1.c2 = s2t2.c1
             left outer join slt2 on slt1.c2 = slt2.c2
-----
-----
-- Same query written more efficiently
-- This enables the optimizer to consider pushing down the outer join of
-- nicknames of the same data source to the remote server
select *
from (select slt1.c1 as c1 from slt1 left outer join slt2 on slt1.c2 = slt2.c1)
     as temp1
```

```
left outer join
(select s2t1.c1 as c1 from s2t1 left outer join s2t2 on s2t1.c2 = s2t2.c1)
  as temp2
on temp1.c1 = temp2.c1
```

Miscellaneous best practices

If a function template is defined and mapped, it should be defined as DETERMINISTIC and NO EXTERNAL ACTION, if possible. This facilitates additional query rewrite possibilities for the DB2 optimizer.

3.4.5 Hardware and network

The speed of the network between DB2 II and the data sources, and the performance characteristics of the data sources' and DB2 II's hardware resources, impact the performance of federated queries.

Network

The network between the DB2 II server and data sources should be 100 Gbit or faster. One Gbit is highly recommended if there are a lot of interactions between DB2 II and data sources during execution of queries or if there are large result sets returned from the data sources to DB2 II.

We also recommend the following:

- ▶ Minimize the number of router hops between the DB2 II server and the data sources.
- ▶ Tune the network packet size for large transfers if there will be large result sets.

Hardware resources at the data sources

Best performance is usually achieved if all or most SQL is pushed down to data sources. Therefore, most of the new workload added by DB2 II users will be created at the data sources.

Hardware resources at DB2 II

If DB2 II functionality is added to a DB2 UDB system, the hardware resources to support the management and access to local tables should be adequate to support the workload of the added DB2 II functionality. This is because most SQL for remote data should be pushed down to data sources for superior performance. Therefore, CPU requirements for the DB2 II functionality should only be a small addition to the CPU requirements to process the SQL on local data. If there are sorts and temporary tables in the DB2 II workload, they can use

the same SORTHEAP and buffer pool for temporary tables for the processing on the local data. The only additional thing to consider is the network adapter speed so that the federated queries have a fast network connection to the remote data sources involved in queries.

The priority in terms of hardware components for DB2 II functionality should typically be:

1. Network adapter
2. Memory for sorts, temp tables, hash/merge scan joins
3. CPU—particularly if there will be large result sets for users
4. Disk configuration—but only for when sorts and temp tables spill from the buffer pool to disk

Performance problem determination scenarios

In this chapter, we discuss some commonly encountered performance problems in a DB2 Information Integrator (II) environment and describe scenarios for identifying and resolving such problems.

The topics covered include:

- ▶ DB2 II hypotheses hierarchy
- ▶ Monitoring best practices
- ▶ Performance problem scenarios

4.1 Introduction

Most IT environments today are complex infrastructures involving heterogeneous network, hardware, and software components organized in multi-tier configurations. Business applications share this complex IT infrastructure that is managed by IT professionals skilled in their particular domains of expertise, for example, network administrators, operating system administrators, Web application server administrators, and database administrators.

Users may experience performance problems for reasons such as network connectivity and bandwidth constraints; system CPU, I/O, and memory constraints; software configuration limitations and constraints; poor systems administration skills; poor application design; and invalid assumptions about the workload.

In 2.4, “Problem determination methodology” on page 37, we discuss a general problem determination methodology, and recommend a hypotheses validation hierarchy that should typically be followed during problem diagnosis of DB2 II applications in general.

For the typical DB2 II environment shown in Figure 4-1 on page 117, the diagnosis process should sequentially eliminate the cause of the problem as follows:

1. Network related—between the client and the Web application server
2. Web application server related—both system (CPU, I/O, memory) and various configuration settings
3. Network related—between the Web application server and the federated database server
4. Federated database server related—system (CPU, I/O, memory), configuration settings, routine DBA maintenance activities such as collecting statistics or creating index specifications, and setting appropriate options for federated objects such as wrappers, servers, and nicknames
5. Network related—between the federated database server and the data source server
6. Application design related—nicknames and SQL

Figure 4-1 on page 117 describes one possible DB2 II environment, having both remote and local clients accessing a DB2 II federated database. While this environment shows the Web application server and the database server on different systems, other application environments may be simpler or more complex depending upon an organization’s unique configuration and application workload requirements.

Important: Following this sequence is strongly recommended in order to ensure that the DBA does not expend needless effort on troubleshooting DB2 II and tuning SQL, when the root cause of the performance problem experienced by the user potentially exists elsewhere. For example, network bandwidth constraints or database performance at a data source server can manifest as erratic or poor response times for a DB2 II federated application even when the DB2 II federated server and application is optimally tuned.

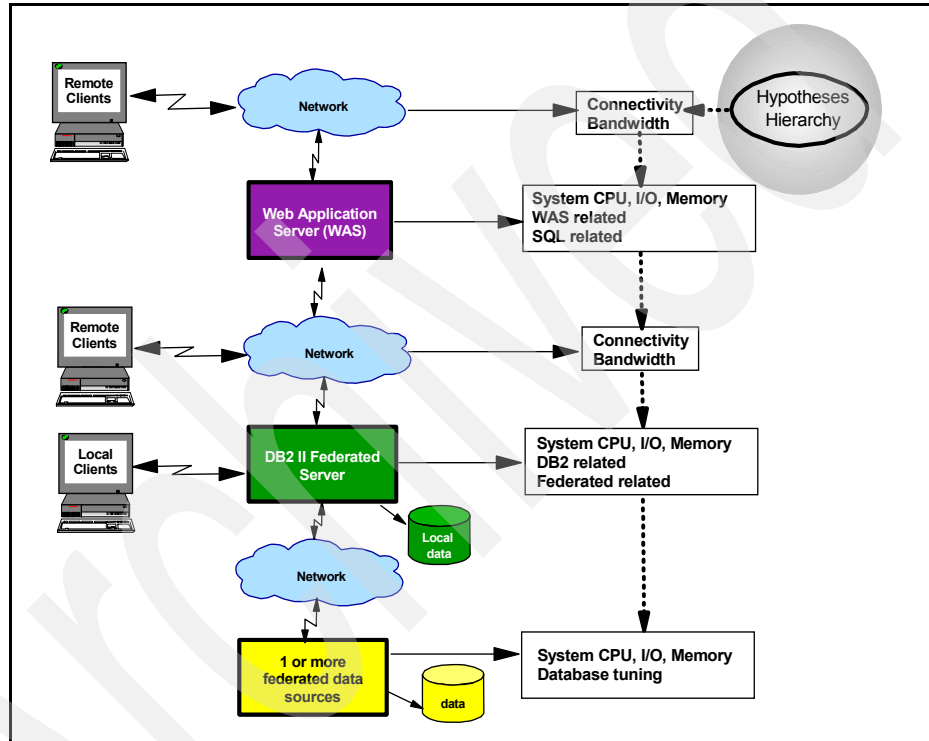


Figure 4-1 A typical DB2 II environment and hypotheses hierarchy

Important: Depending upon the triggering event of the performance problem, it may be possible to skip certain hypotheses validation altogether; for example, when an explicit alert about a nickname being invalid is the triggering event, one can ignore hypotheses such as network connectivity and bandwidth constraints, Web application server constraints, and CPU and I/O constraints (in both the Web application server and the DB2 federated database server) as a potential cause of the problem. More on this later.

Table 4-1 provides a very high-level overview of the resource constraint conditions associated with the components that an application utilizes in the execution of its functions, along with tools that can be used to investigate them.

Table 4-1 Typical problem areas associated with DB2 II performance

Component	Resource constraint conditions	Tools
Network	<ul style="list-style-type: none"> ► Connectivity and bandwidth 	ip ping, connect, db2 ping, ftp
System	<ul style="list-style-type: none"> ► CPU utilization ► I/O utilization and performance ► Memory paging 	<ul style="list-style-type: none"> ► vmstat, nmon, uptime, ruptime, Task Manager, Performance Monitor ► iostat, nmon, Task Manager, Performance Monitor, filemon ► vmstat, nmon, svmon, ipcs, Task Manager, Performance Monitor, Memory Visualizer
Web application server	<ul style="list-style-type: none"> ► System resource constraints ► Connections to DB2 ► Configuration parameters 	<ul style="list-style-type: none"> ► Same as system ► Tivoli® Performance Viewer ► WebSphere tools
DB2 federated server	<ul style="list-style-type: none"> ► System resource constraints ► DB2 resource constraints ► Federated definitions ► DB2 application design 	<ul style="list-style-type: none"> ► Same as system ► DB2 system monitor DB2 Control Center (Health Center) ► DB2 commands and tools, explain information
Federated data source	<ul style="list-style-type: none"> ► System resource constraints ► Database performance tuning 	Varies by operating system platform and type of database

Please refer to the appropriate manuals for information about the tools mentioned in Table 4-1.

Important: In most cases, the DBA has no jurisdiction over monitoring and tuning network, system, and Web application server performance drivers, since they are the responsibility of the appropriate administrator.

The objective here is to make the DBA aware of the critical impact on the performance of their DB2 II environment by the various network, operating system, and Web application server performance drivers, so that he may be in a position to negotiate more effectively with the appropriate administrators responsible for these performance drivers.

In the following sections we focus on:

- ▶ Resource constraint conditions related to DB2 Information Integrator, and recommend the hypotheses hierarchy to adopt for problem diagnosis
- ▶ Monitoring best practices for routine, online/event, and exception monitoring of a DB2 II environment
- ▶ Performance problem scenarios

Note: We used DB2 UDB ESE Version 8.2 and DB2 II Version 8.2 in all the scenarios discussed in the following section.

4.2 DB2 II hypotheses hierarchy

Once the network and systems have been tentatively eliminated as the potential source of the performance problems, we need to narrow the focus to the federated database server itself. However, within the federated database server itself, there is a definite hierarchy of hypotheses validation that must be adopted for effective problem diagnosis, as shown in Figure 4-2 on page 120.

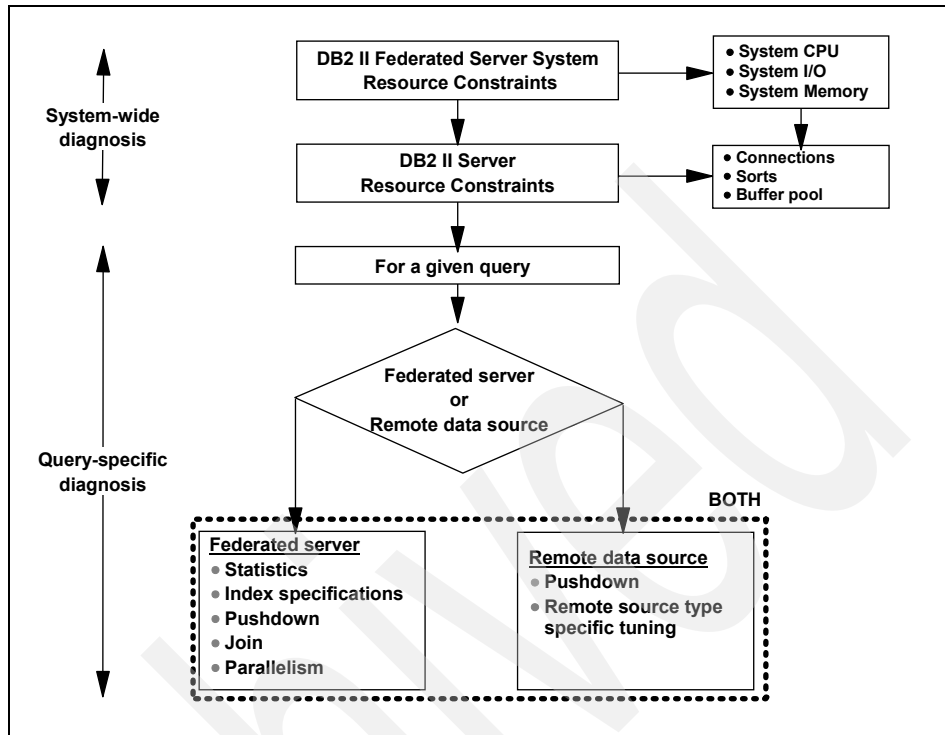


Figure 4-2 DB2 II hypotheses hierarchy

Important: Figure 4-2 assumes that the DB2 II federated database server is isolated on a separate system and does not have any local DB2 tables that participate in user queries.

If local tables are involved, then all the considerations of tuning a DB2 system would apply in addition. A discussion of tuning local tables is beyond the scope of this publication. Please refer to the redbook *DB2 UDB ESE V8 non-DPF Performance Guide for High Performance OLTP and BI*, SG24-6432, for a complete discussion of this subject.

The overall flow is as follows:

- ▶ Ensure that the federated database server system is not constrained on resources such as CPU, I/O, and memory; this is a system-wide diagnosis.
- ▶ Next ensure that system-wide DB2 II system resources are not being constrained in any way; this includes connection limits being tripped, and the sort and buffer pool (temporary table space) are not constrained in any way. This is also a system-wide diagnosis.

- After these system-wide items are tentatively eliminated as potential causes of performance problems, we need to identify and focus on the specific query experiencing the performance problem.

Note: The approach used to pinpoint the problem query depends upon whether the query is currently running.

- If the query is not currently running:
 - You may check the contents of the global dynamic cache to determine long-running queries and try to associate them with the application experiencing performance problems.
 - You may want to ask the user if the query can be rerun so that the problem can be reproduced. This may be the easiest way and provide you with the best diagnostics to analyze the problem. The CLI trace may be enabled if appropriate for more detailed diagnostics information.
- If the query is currently running, there are tools such as the snapshot monitor (such as the command **get snapshot for all remote_applications**), DB2 Performance Expert, and other third-party monitoring tools that can assist in identifying currently executing queries and the resources they are consuming.

If such tools are not available, a possible approach is to take a snapshot of all applications that are currently executing against the database, and use your knowledge of the application names, connection times, and statement text to identify the poorly performing query.

This involves determining whether the query's performance problem is at the remote data source or on the federated server side, or both. This is primarily determined by a combination of DB2 EXPLAIN output for the query, the metrics obtained from the dynamic cache using the **get snapshot for dynamic sql on <dbname>** command, and the **get snapshot for remote_applications on <dbname>** or **get snapshot for all remote_applications** command.

Important: As mentioned earlier, depending upon the triggering event for performance problem diagnosis, it may be possible to skip certain hypotheses validation altogether and enter the hypotheses hierarchy at a lower point. This idea is highlighted in Figure 4-3. For instance, widespread user complaints involving different applications would tend to point to a system-wide performance problem (entering the DB2 II hypotheses hierarchy at the top by beginning with the network), while pointed complaints from few users of a specific application/query would tend to point to performance problems with that particular application/query. In the latter case, one could conceivably ignore the system-wide hypotheses (entering the DB2 II hypotheses hierarchy at the top by beginning with the network) and focus on identifying the problem query (entering the DB2 II hypotheses hierarchy lower down) and performing diagnosis on it to establish the root cause.

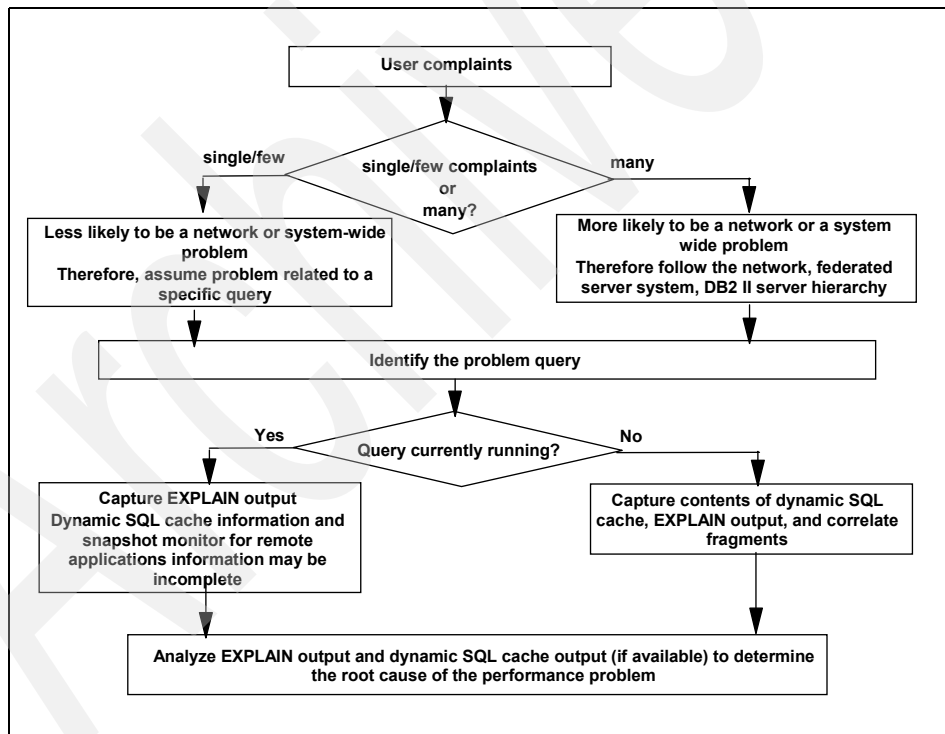


Figure 4-3 Triggering event determines entry into DB2 II hypotheses hierarchy

- ▶ If the performance problem is determined to be primarily at the federated server, then a number of possible items need to be investigated, including:
 - Current statistics about nicknames in the federated global system catalog.
 - Current index specifications on nicknames in the federated global system catalog.
 - Degree (full, partial, or none) of pushdown of predicates down to the remote data sources. If partial or none, one needs to further investigate the cause, which may include the syntax of the query, or the server or column nickname options.
 - Type of join chosen by the DB2 optimizer; for example, a hash join may seem to be more appropriate for the query even though the DB2 optimizer chose a nested loop join.
 - Inhibition of intra-partition and inter-partition parallelism, which is appropriate when local data access is involved in the query. One of the causes could be the default setting of the INTRA_PARALLEL database manager configuration parameter. Check the setting of wrapper option DB2_FENCED. Full exploitation of parallelism when nicknames and local tables are joined requires that wrapper option DB2_FENCED be set to 'Y'. This is also required for inter-partition parallelism to be enabled for nickname-only queries (use of Computational Partition Groups) for processing of large intermediate result sets and non-pushed-down SQL operations.
- ▶ If the performance problem is determined to be primarily at the remote data source, then appropriate tuning strategies unique to the type of remote data source need to be adopted by the remote administrator.

However, in the case of remote relational databases, one also needs to investigate the degree of pushdown of predicates to determine if any of the DB2 II configuration options (related to the DB2 database manager, DB2 II server, or DB2 II nickname columns) need review or whether the query syntax itself is inhibiting pushdown.
- ▶ If the performance problem is at both the federated server and the remote data source, then all the items mentioned should be investigated together.

Each of the items in the hierarchy shown in Figure 4-2 on page 120 are described in further detail in the following subsections.

4.2.1 DB2 II federated database server resource constraints

The system on which the DB2 II federated database server is running needs to be monitored to ensure that CPU, I/O, and memory consumption is within normal bounds before validating hypotheses further down in the hierarchy.

In some cases, this monitoring of the system resources in general, and DB2 processes in particular, may highlight potential problem areas that need further investigation lower down in the hierarchy to pinpoint the problem; for example, high I/O utilization may indicate excessive contention due to poor placement of highly active table spaces on the same drive.

Note: Checking whether DB2 processes are running is a special case of system-level resource consumption checking.

The AIX `ps -ef | grep db2` command and the Windows Task Manager can be used to identify DB2 processes.

4.2.2 DB2 II resource constraints

Certain DB2 database manager and database configuration settings may result in federated applications suffering response time problems due to the following:

- ▶ Connection constraints
- ▶ Sorting constraints
- ▶ Locking constraints
- ▶ Buffer pool constraints
- ▶ Cache size constraints such as catalog and package
- ▶ Miscellaneous constraints such as enabling intra-partition parallelism

Important: Table 3-1 on page 55 highlights the performance impact of these constraints in a federated environment depending upon whether it is a dedicated federated server or a collocated federated server environment.

These constraints are *not* ordered by the impact they have on performance, but by the sequence in which problem diagnosis is recommended.

In many cases, problem diagnosis is a holistic affair where a particular problem is best diagnosed by combining monitoring results of multiple events and entities.

Important: This section does *not* discuss every monitoring and tuning knob available in DB2 to manage the performance of the DB2 environment.

The purpose of this section is to promote a top-down discipline to the process of problem diagnosis by discussing some of the more important DB2 resource constraints that impact DB2 II performance.

In practice, the impact of a particular resource constraint on overall DB2 II performance depends a great deal upon the nature and priorities of the application workload and the resources available for its execution. For example, with a read-only data warehousing type workload, locking constraints are likely to play an insignificant role in overall DB2 II performance, while sorting constraints will probably have considerable performance impact.

The following subsections identify many of the key database manager configuration and database configuration parameters associated with each of these constraints, and describe some of the monitoring elements that should be used to configure them optimally.

Note: Database manager configuration settings can be viewed by issuing the following command:

```
db2 get dbm cfg
```

Database configuration settings can be viewed by issuing the following command:

```
db2 get db cfg for <database_name>
```

The monitoring elements are reported in the snapshot monitor, and may represent water marks, counters, time, and gauges.

Attention: When the monitoring elements are *not* high water marks, they should be sampled at specific intervals over an extended period of time to get a realistic view of system usage.

In some cases, it may be necessary to reset counters at the start of the monitoring interval to get an accurate view of activity.

The DB2 Health Center also provides capabilities to monitor a number of key performance elements and generate alerts when values in these performance monitoring elements exceed or trip user-defined thresholds.

Connection constraints

Connection problems are generally manifested as messages generated by an application when a connection exception is returned to the application. The following database manager configuration and database configuration parameters can constrain the number of connections permitted:

- ▶ Database manager configuration parameters
 - max_connections
 - maxagents
 - maxcagents
 - max_coordagents
- ▶ Database configuration parameters
 - maxappls

To determine whether connection problems are being experienced, the snapshot monitor should be invoked during the appropriate monitor interval, and relevant fields checked, as follows:

- ▶ For the database manager configuration parameters MAX_CONNECTIONS, MAXAGENTS, MAXCAGENTS, and MAX_COORDAGENTS, relevant snapshot contents are shown in Example 4-1.

Example 4-1 dbm snapshot for connections

```
db2 => get snapshot for dbm
...
Remote connections to db manager          = 0
Remote connections executing in db manager = 0
Local connections                        = 4
Local connections executing in db manager = 0
Active local databases                   = 1

High water mark for agents registered      = 5
High water mark for agents waiting for a token = 0
Agents registered                         = 5
Agents waiting for a token                 = 0
Idle agents                              = 0
....
Agents assigned from pool                  = 2
Agents created from empty pool             = 7
Agents stolen from another application     = 0
High water mark for coordinating agents   = 5
Max agents overflow                        = 0
Hash joins after heap threshold exceeded  = 0
.....
```

Check the following values:

- Sum of (**Remote connections to db manager** + **Local connections**) should be less than MAX_CONNECTIONS.

Note: These values are *not* high water marks, and should therefore be sampled at specific intervals over an extended period of time and representative intervals to get a realistic view of system usage.

If this value is the same as the MAX_CONNECTIONS, then it is likely that some database connection requests were rejected.

- **High water mark for agents registered** should be less than MAXAGENTS.

If this value is the same as the MAXAGENTS, then it means that connections may have failed.

- **High water mark for agents waiting for a token** should be equal to zero in unconstrained systems.

If this value is the same as the MAXCAGENTS, then it means that certain applications had to wait since a transaction cannot be initiated without getting a token.

- **High water mark for coordinating agents** should be less than MAX_COORDAGENTS.

If this value is the same as the MAX_COORDAGENTS, then it means that agent creations may have failed.

- For the database configuration parameter MAXAPPLS, relevant snapshot contents are shown in Example 4-2.

Example 4-2 db snapshot for connections

```
db2 => get snapshot for all on sample
...
High water mark for connections           = 3
Application connects                     = 3
Secondary connects total                  = 0
Applications connected currently          = 3
Appls. executing in db manager currently = 0
Agents associated with applications       = 3
Maximum agents associated with applications= 3
Maximum coordinating agents              = 3
...
```

Check that the **High water mark for connections** is less than MAXAPPLS, unless MAXAPPLS is specified as being AUTOMATIC.

If this value is the same as the MAXAPPLS parameter, then it is likely that some database connection requests were rejected.

Sorting constraints

Sorting problems are generally manifested as poor response times for the application, and the occasional application error message when sort heap hard limits are exceeded.

The following database manager configuration and database configuration parameters can impact sort performance:

- ▶ Database manager configuration parameters
 - SHEAPTHRES
- ▶ Database configuration parameters
 - SHEAPTHRES_SHR
 - SORTHEAP

To determine whether sorting problems are being experienced, the snapshot monitor should be invoked during the appropriate monitor interval, and relevant fields checked as follows:

- ▶ For the database manager configuration parameter SHEAPTHRES, relevant snapshot contents are shown in Example 4-3.

Example 4-3 dbm snapshot for sorting

```
db2 => get snapshot for dbm
...
Private Sort heap allocated           = 0
Private Sort heap high water mark    = 277
Post threshold sorts                 = 0
Piped sorts requested                 = 10
Piped sorts accepted                  = 10
...
Agents assigned from pool             = 2
Agents created from empty pool        = 7
Agents stolen from another application = 0
High water mark for coordinating agents = 5
Max agents overflow                   = 0
Hash joins after heap threshold exceeded = 0
...
```

Check the following values:

- **Private Sort heap high water mark** should be less than SHEAPTHRES.

If this value is greater than or equal to SHEAPTHRES, then it means that the sorts are not getting the full sort heap as specified by the SORTHEAP database configuration parameter.

- **Post threshold sorts** should be very small if not zero.

When post threshold sorts occur, the database manager allocates a smaller sort heap than that specified by the SORTHEAP database configuration parameter. Subsequent sort heap allocations are reduced even further until the total amount of sort heap in use falls below the amount specified for SHEAPTHRES. This situation causes a serious degradation in database performance and should be avoided.

Note: This value is *not* a high water mark, and should therefore be sampled at specific intervals over an extended period of time and representative intervals to get a realistic view of such occurrences.

- Difference between **Piped sorts requested** and **Piped sorts accepted** should be very small if not zero.

If this value is high, then it means that piped sorts are being rejected because the sort heap threshold (SHEAPTHRES) would be exceeded when the sort heap is allocated for the sort.

Note: Here too, these values are *not* high water marks, and should therefore be sampled at specific intervals over an extended period of time and representative intervals to get a realistic view of such occurrences.

- **Hash joins after heap threshold exceeded** should be very small if not zero.

If this value is non-zero, then it means that a hash join heap request was limited because of the sort heap threshold (SHEAPTHRES) being exceeded. This value should be used in conjunction with **Hash join overflows** to calculate the percentage of such occurrences.

Note: This value is *not* a high water mark, and should therefore be sampled at specific intervals over an extended period of time and representative intervals to get a realistic view of such occurrences.

- For the database configuration parameters SHEAPTHRES_SHR and SORTHEAP, relevant snapshot contents are shown in Example 4-4 on page 130.

```
db2 => get snapshot for all on sample
...
Total Private Sort heap allocated          = 0
Total Shared Sort heap allocated          = 0
Shared Sort heap high water mark          = 0
Total sorts                             = 1
Total sort time (ms)                      = Not Collected
Sort overflows                           = 1
Active sorts                             = 0
...
```

Check the following values:

- **Shared Sort heap high water mark** should be less than SHEAPTHRES_SHR.

If this value is equal to SHEAPTHRES_SHR, then it means that the total amount of database shared memory that can be used for sorting at any one time has been exceeded, and subsequent sorts will fail with an SQL0955C message.

- **Sort overflows** should be very small if not zero.

A non-zero value indicates that sorts ran out of sort heap (SORTHEAP) and had to overflow to disk. This value should be used in conjunction with **Total sorts** to calculate the percentage of sorts that overflowed to disk.

Note: This value is *not* a high water mark, and should therefore be sampled at specific intervals over an extended period of time and representative intervals to get a realistic view of such occurrences.

Locking constraints

Locking problems are generally manifested as poor response times, and the occurrences of deadlocks and time outs. Certain locking problems such as lock escalations and deadlocks are also reported in the *db2diag.log*.

Note: Locking constraints are unlikely to cause performance problems in a DB2 II environment since the activity tends to be predominantly read only using the cursor stability isolation level.

However, concurrency issues may arise with local data and MQTs, and the isolation level passed to the remote data source. The likelihood of concurrency issues with a federated query are more likely to occur with backend data sources than the federated server. This section describes DB2 locking related parameters as it relates to local data and MQTs, and remote DB2 data sources.

The following database configuration parameters can severely impact the amount of concurrency and throughput achievable:

- ▶ dlchktme
- ▶ locklist
- ▶ locktimeout
- ▶ maxlocks

To determine whether locking problems are being experienced, the snapshot monitor should be invoked during the appropriate monitor interval, and relevant fields checked, as shown in Example 4-5.

Example 4-5 db snapshot for locking

```
db2 => get snapshot for all on sample
...
Locks held currently                = 1
Lock waits                          = 0
Time database waited on locks (ms) = 0
Lock list memory in use (Bytes)     = 540
Deadlocks detected                  = 0
Lock escalations                    = 0
Exclusive lock escalations          = 0
Agents currently waiting on locks   = 0
Lock Timeouts                       = 0
...
```

Check the following values:

- ▶ The ratio of **Time waited on locks** and **Lock waits** is a measure of the average duration of each lock wait, and should be subsecond or a few seconds only.

If this number is high, it could be due to applications holding too many locks, for extended durations, or lock escalations. It is also possible that extended lock wait durations may point to a problem with the LOCKTIMEOUT value.
- ▶ **Lock escalations** and **Exclusive lock escalations** should be kept to a minimum.

If a large number of escalations are seen over a short interval, then it could mean that the LOCKLIST and/or MAXLOCKS values are too small. These values should be evaluated in conjunction with **Deadlocks detected**, **Lock waits** and **Time waited on locks**.

Note: Further detailed information about lock escalations can be obtained from the *db2diag.log*.

- **Deadlocks detected** and **Lock timeouts** should be kept to a minimum.

If a large number of deadlocks and lock timeouts are seen over a short interval, then it could mean that the DLCHKTIME value is too large and the LOCKTIMEOUT value is too small. Here again, these values should be evaluated in conjunction with **Lock escalations**, **Exclusive lock escalations**, **Lock waits** and **Time waited on locks**.

Note: In all the above cases, the monitoring elements are counters or gauges representing a specific point in time, and should therefore be sampled at peak or heavy workload intervals to ascertain whether locking problems exist. This should involve resetting the counters at the start of the monitoring interval.

Buffer pool constraints

Buffer pool problems are generally manifested as poor response times as a result of increased synchronous I/Os or operating system paging. The key metric to measure here is the hit ratio achieved—the higher the better.

Buffer pools are defined via the CREATE BUFFERPOOL SQL statement, and are allocated at database activation. Buffer pools are used by tables, indexes, and sort temporary table spaces.

Buffer pool information is typically gathered at a table space level, but the facilities of the database system monitor can roll this information up to the buffer pool and database levels.

Attention: In DB2 II environments with no local table access, the buffer pool of interest is the one associated with the temporary tablespaces used in sorting. This buffer pool should be tuned for optimal performance when federated queries involve significant amounts of sorting.

To determine whether buffer pool problems are being experienced, the snapshot monitor should be invoked during the appropriate monitor interval, and relevant fields checked, as shown in Example 4-6.

Example 4-6 db snapshot for buffer pools

```
$ db2 "get snapshot for all on dtw"
.....
Buffer pool data logical reads      = 5160
Buffer pool data physical reads    = 1485
Buffer pool data writes             = 0
Buffer pool index logical reads    = 18371
Buffer pool index physical reads    = 3064
Total buffer pool read time (ms)    = 69774
```

Total buffer pool write time (ms)	= 0
Asynchronous pool data page reads	= 0
Asynchronous pool data page writes	= 0
Buffer pool index writes	= 0
Asynchronous pool index page reads	= 16
Asynchronous pool index page writes	= 0
Total elapsed asynchronous read time	= 268
Total elapsed asynchronous write time	= 0
Asynchronous read requests	= 0
Direct reads	= 236
Direct writes	= 0
Direct read requests	= 58
Direct write requests	= 0
Direct reads elapsed time (ms)	= 483
Direct write elapsed time (ms)	= 0
Database files closed	= 0
Data pages copied to extended storage	= 0
Index pages copied to extended storage	= 0
Data pages copied from extended storage	= 0
Index pages copied from extended storage	= 0
Unread prefetch pages	= 0
Vectored I/Os	= 0
Pages from vectored I/Os	= 0
Block I/Os	= 0
Pages from block I/Os	= 0
Physical page maps	= 0

- Compute the total buffer pool hit ratio as:

$$(1 - ((\text{Buffer pool data physical reads} + \text{Buffer pool index physical reads}) / (\text{Buffer pool data logical reads} + \text{Buffer pool index logical reads}))) * 100\%$$
- Compute the index hit ratio as:

$$(1 - ((\text{Buffer pool index physical reads}) / (\text{Buffer pool index logical reads}))) * 100\%$$
- Writes to the buffer pool associated with the temporary tablespace (**Buffer pool data writes** and **Asynchronous pool data page writes**) indicate buffers being written to disk, and may suggest increasing the number of buffers to eliminate disk I/O.

Note: In this case, the monitoring elements are counters representing a specific point in time, and should therefore be sampled at peak or heavy workload intervals to ascertain whether buffer pool problems exist — this should involve resetting the counters at the start of the monitoring interval.

Cache size constraints

There are several caches in DB2, and problems with their size are generally manifested as poor response times as a result of a increased synchronous I/Os or operating system paging.

Note: There are several database manager and database caches that can be specified such as ASLHEAPSZ, AUDIT_BUF_SZ, JAVA_HEAP_SZ, QUERY_HEAP_SZ, and APPLHEAPSZ to name a few. The objective is to determine if these caches are too small or too big. Each one of these has slightly different monitor elements to look at as described in the *IBM DB2 UDB System Monitor Guide and Reference*, SC09-4847.

We focus here on CATALOGCACHE_SZ and PCKCACHESZ. The key metrics to assess the efficiency of these caches are hit ratios, overflows, and high water marks.

To determine whether CATALOGCACHE_SZ and PCKCACHESZ problems are being experienced, the snapshot monitor should be invoked during the appropriate monitor interval, and relevant fields checked as shown in Example 4-7.

Example 4-7 db snapshot for catalogcache_sz and pckcachesz

```
db2 => get snapshot for all on sample
...
Package cache lookups           = 4
Package cache inserts           = 2
Package cache overflows         = 0
Package cache high water mark (Bytes) = 118968
Application section lookups     = 6
Application section inserts     = 2

Catalog cache lookups           = 13
Catalog cache inserts           = 5
Catalog cache overflows         = 0
Catalog cache high water mark   = 0
...
```

The following considerations apply to some of the fields described in Example 4-2 on page 127:

- ▶ The hit ratio for the package cache is computed as follows:
$$(1 - (\text{Package cache inserts} / \text{Package cache lookups})) * 100\%$$
- ▶ **Package cache overflows** should be kept as zero.

If this number is non-zero, then it means that a spillover has occurred into other heaps such as LOCKLIST, thereby resulting in unnecessary lock escalation and general performance degradation.

- **Package cache high water mark (Bytes)** represents the largest size reached by the package cache, and can be used to determine the value for the PCKCACHESZ database configuration parameter to avoid cache overflows.

Note: Except for the **Package cache high water mark (Bytes)** monitoring element, the other elements reported are counters representing a specific point in time, and should therefore be sampled at peak or heavy workload intervals to ascertain whether package cache size problems exist. This should involve resetting the counters at the start of the monitoring interval.

Attention: The same considerations as discussed for PCKCACHESZ also apply to CATALOGCACHE_SZ.

Miscellaneous constraints

There are a host of other database manager and database configuration parameters that can impact the performance of DB2 II applications. Details of their individual performance impact is beyond the scope of this publication. Please refer to the redbook *DB2 UDB ESE V8 non-DPF Performance Guide for High Performance OLTP and BI*, SG24-6432, for a complete discussion of this subject.

Attention: In DB2 II environments with no local table access, the following list of database manager and database configuration parameters are unlikely to impact DB2 Information Integrator performance. However, federated environments with local data access need to be tuned effectively for superior performance, and the following parameters are part of the tuning knobs available to achieve this end.

Some of the parameters of interest are as follows:

- Database manager configuration parameters
 - intra_parallel
 - maxfilop and maxtotfilop
 - rqrioblk
 - num_poolagents
- Database configuration parameters
 - chngpgs_thresh

- logbufsz
- num_iocleaners
- num_ioservers
- Registry and environment variables
 - DB2_PARALLEL_IO
 - DB2MEMDISCLAIM
 - DB2MEMMAXFREE
 - DB2NTMEMSIZE
 - DB2_PINNED_BP

4.2.3 Federated server or remote data source

Determining whether the resources consumed for a query is predominantly in the federated server, or at the remote data source, or somewhat equally distributed between them requires reviewing the query in the dynamic SQL snapshot output and EXPLAIN output of the query.

Example 4-8 is an example of snapshot output using the **get snapshot for dynamic sql ...** command, in which some of the key fields are highlighted. Example 4-9 on page 138 shows **db2exfmt** output for the SQL query shown in Example 4-8.

Example 4-8 Dynamic SQL snapshot

```
$db2 get snapshot for dynamic sql on fedserv
```

Dynamic SQL Snapshot Result

Database name	= FEDSERV
Database path	= /data1/kawa/kawa/NODE0000/SQL00001/
Number of executions	= 374
Number of compilations	= 1
Worst preparation time (ms)	= 77
Best preparation time (ms)	= 77
Internal rows deleted	= 0
Internal rows inserted	= 0
Rows read	= 3578
Internal rows updated	= 0
Rows written	= 0
Statement sorts	= 374
Statement sort overflows	= 0
Total sort time	= 322696
Buffer pool data logical reads	= 0
Buffer pool data physical reads	= 0
Buffer pool temporary data logical reads	= 0

```

Buffer pool temporary data physical reads = 0
Buffer pool index logical reads = 0
Buffer pool index physical reads = 0
Buffer pool temporary index logical reads = 0
Buffer pool temporary index physical reads = 0
Total execution time (sec.ms) = 16984.947355
Total user cpu time (sec.ms) = 2009.080000
Total system cpu time (sec.ms) = 85.490000
Statement text = SELECT O_ORDERPRIORITY,
COUNT(*) AS ORDER_COUNT FROM ORA.ORDERS WHERE O_ORDERKEY
BETWEEN 1 AND 1000000 AND EXISTS (SELECT * FROM
DB2.LINEITEM WHERE L_ORDERKEY = O_ORDERKEY
AND L_COMMITDATE < L_RECEIPTDATE ) GROUP BY
O_ORDERPRIORITY ORDER BY O_ORDERPRIORITY

```

.....lines have been removed.....

```

Number of executions = 374
Number of compilations = 374
Worst preparation time (ms) = 0
Best preparation time (ms) = 0
Internal rows deleted = 0
Internal rows inserted = 0
Rows read = 236474590
Internal rows updated = 0
Rows written = 0
Statement sorts = 0
Buffer pool data logical reads = 0
Buffer pool data physical reads = 0
Buffer pool temporary data logical reads = 0
Buffer pool temporary data physical reads = 0
Buffer pool index logical reads = 0
Buffer pool index physical reads = 0
Buffer pool temporary index logical reads = 0
Buffer pool temporary index physical reads = 0
Total execution time (sec.ms) = 13144.507706
Total user cpu time (sec.ms) = 0.000000
Total system cpu time (sec.ms) = 0.000000
Statement text = SELECT AO."L_ORDERKEY" FROM
"TPCD"."LINEITEM" AO WHERE (AO."L_COMMITDATE" < AO."L_RECEIPTDATE") AND
(AO."L_ORDERKEY" <= 1000000) AND (1 <= AO."L_ORDERKEY") ORDER BY 1 ASC,
AO."L_COMMITDATE" ASC, AO."L_RECEIPTDATE" ASC FOR READ ONLY

```

.....lines have been removed.....

```

Number of executions = 374
Number of compilations = 374
Worst preparation time (ms) = 0
Best preparation time (ms) = 0

```

```

Internal rows deleted          = 0
Internal rows inserted        = 0
Rows read                    = 93500000
Internal rows updated         = 0
Rows written                  = 0
Statement sorts               = 0
Buffer pool data logical reads = 0
Buffer pool data physical reads = 0
Buffer pool temporary data logical reads = 0
Buffer pool temporary data physical reads = 0
Buffer pool index logical reads = 0
Buffer pool index physical reads = 0
Buffer pool temporary index logical reads = 0
Buffer pool temporary index physical reads = 0
Total execution time (sec.ms) = 2724.553063
Total user cpu time (sec.ms)   = 0.000000
Total system cpu time (sec.ms) = 0.000000
Statement text              = SELECT AO."O_ORDERKEY",
AO."O_ORDERPRIORITY", AO.ROWID FROM "IITEST"."ORDERS" AO WHERE (1 <=
AO."O_ORDERKEY") AND (AO."O_ORDERKEY" <= 1000000) ORDER BY 1 ASC

```

Example 4-8 on page 136 shows the user-entered SQL statement as well as two remote SQL fragments, which tends to look like “SELECT A?.”colname”....FROM...”.

Attention: In practice there will be many user-entered SQL as well as remote SQL fragments listed in the output of the dynamic SQL snapshot command.

However, there is no direct mechanism to link the remote SQL fragments in the dynamic cache with their corresponding user SQL statement.

In order to determine the relationship between a user-entered SQL statement and the remote SQL fragments associated with it, you need to EXPLAIN the user-entered query and determine the remote SQL fragment(s) text identified in the RMTQTXT field of the SHIP operator (6 and 10), as shown in Example 4-9.

Important: The remote SQL fragment text in the RMTQTXT field of the SHIP operator of **db2exfmt** output contains the names of the remote data source tables/views, and *not* the nicknames referencing these remote objects in the use query.

Example 4-9 Sample db2exfmt output with a SHIP operator and RMTQTXT field

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM

***** EXPLAIN INSTANCE *****

DB2_VERSION: 08.02.0
SOURCE_NAME: TOOL1E00
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-07-02-19.24.26.071589
EXPLAIN_REQUESTER: KAWA

Database Context:

Parallelism: None
CPU Speed: 4.723442e-07
Comm Speed: 100
Buffer Pool size: 78000
Sort Heap size: 20000
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 1020

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 1 -----

QUERYNO: 1
QUERYTAG:
Statement Type: Select
Updatable: No
Deletable: No
Query Degree: 1

Original Statement:

SELECT O_ORDERPRIORITY, COUNT(*) AS ORDER_COUNT
FROM ORA.ORDERS
WHERE O_ORDERKEY BETWEEN 1 AND 1000000 AND EXISTS
(SELECT *

```

FROM DB2.LINEITEM
WHERE L_ORDERKEY = O_ORDERKEY AND L_COMMITDATE < L_RECEIPTDATE )
GROUP BY O_ORDERPRIORITY
ORDER BY O_ORDERPRIORITY

```

Optimized Statement:

```

-----
SELECT Q4.$C0 AS "O_ORDERPRIORITY", Q4.$C1 AS "ORDER_COUNT"
FROM
  (SELECT Q3.$C0, COUNT(*)
   FROM
     (SELECT DISTINCT Q2.O_ORDERPRIORITY, Q2.$P-ROWID$
      FROM DB2.LINEITEM AS Q1, ORA.ORDERS AS Q2
      WHERE (Q1.L_ORDERKEY = Q2.O_ORDERKEY) AND (Q1.L_COMMITDATE <
        Q1.L_RECEIPTDATE) AND (Q2.O_ORDERKEY <= 1000000) AND (1 <=
        Q2.O_ORDERKEY)) AS Q3
   GROUP BY Q3.$C0) AS Q4
ORDER BY Q4.$C0

```

Access Plan:

```

-----
Total Cost: 222926
Query Degree:1

```

```

Rows
RETURN
( 1)
Cost
I/O
|
5
GRPBY
( 2)
222926
16810.7
|
250002
TBSCAN
( 3)
222896
16810.7
|
250002
SORT
( 4)
222762
16810.7
|

```

```

250002
MSJOIN
( 5)
222162
16810.7
/---+---\
250002      1.99876
SHIP        FILTER
( 6)        ( 9)
37714.3     184277
9486.34     7324.35
|           |
1.5e+07     499694
NICKNM: ORA SHIP
ORDERS      ( 10)
            184277
            7324.35
            |
            5.99861e+07
            NICKNM: DB2
            LINEITEM

```

1) RETURN: (Return Result)

Cumulative Total Cost: 222926
Cumulative CPU Cost: 4.89221e+09
Cumulative I/O Cost: 16810.7
Cumulative Re-Total Cost: 222327
Cumulative Re-CPU Cost: 3.62365e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 222794
Estimated Bufferpool Buffers: 0
Remote communication cost:397356

Arguments:

```

-----
BLDLEVEL: (Build level)
DB2 v8.1.1.64 : s040509
ENVVAR : (Environment Variable)
DB2_EXTENDED_OPTIMIZATION = ON
STMTHEAP: (Statement heap size)
8192

```

Input Streams:

```

-----
9) From Operator #2

```

Estimated number of rows: 5
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q5.0_ORDERPRIORITY(A)+Q5.ORDER_COUNT

2) GRPBY : (Group By)

Cumulative Total Cost: 222926
Cumulative CPU Cost: 4.89221e+09
Cumulative I/O Cost: 16810.7
Cumulative Re-Total Cost: 222327
Cumulative Re-CPU Cost: 3.62364e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 222794
Estimated Bufferpool Buffers: 0
Remote communication cost:397356

Arguments:

AGGMODE : (Aggregation Mode)
COMPLETE
GROUPBYC: (Group By columns)
TRUE
GROUPBYN: (Number of Group By columns)
1
GROUPBYR: (Group By requirement)
1: Q3.0_ORDERPRIORITY
ONEFETCH: (One Fetch flag)
FALSE

Input Streams:

8) From Operator #3

Estimated number of rows: 250002
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q3.0_ORDERPRIORITY(A)

Output Streams:

9) To Operator #1

Estimated number of rows: 5
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q5.0_ORDERPRIORITY(A)+Q5.ORDER_COUNT

3) TBSCAN: (Table Scan)

Cumulative Total Cost: 222896
Cumulative CPU Cost: 4.82971e+09
Cumulative I/O Cost: 16810.7
Cumulative Re-Total Cost: 222297
Cumulative Re-CPU Cost: 3.56114e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 222762
Estimated Bufferpool Buffers: 0
Remote communication cost:397356

Arguments:

MAXPAGES: (Maximum pages for prefetch)
ALL
PREFETCH: (Type of Prefetch)
NONE
SCANDIR : (Scan Direction)
FORWARD

Input Streams:

7) From Operator #4

Estimated number of rows: 250002
Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:

+Q2.0_ORDERPRIORITY(A)+Q2.\$RID\$(A)
+Q2.\$P-ROWID\$(A)

Output Streams:

8) To Operator #2

Estimated number of rows: 250002

Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q3.O_ORDERPRIORITY(A)

4) SORT : (Sort)
Cumulative Total Cost: 222762
Cumulative CPU Cost: 4.54445e+09
Cumulative I/O Cost: 16810.7
Cumulative Re-Total Cost: 222162
Cumulative Re-CPU Cost: 3.27589e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 222762
Estimated Bufferpool Buffers: 16811.9
Remote communication cost:397356

Arguments:

DUPLWARN: (Duplicates Warning flag)
TRUE
NUMROWS : (Estimated number of rows)
250002
ROWWIDTH: (Estimated width of rows)
44
SORTKEY : (Sort Key column)
1: Q2.O_ORDERPRIORITY(A)
SORTKEY : (Sort Key column)
2: Q2.\$RID\$(A)
SORTKEY : (Sort Key column)
3: Q2.\$P-ROWID\$(A)
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
TRUE

Input Streams:

6) From Operator #5

Estimated number of rows: 250002
Number of columns: 5
Subquery predicate ID: Not Applicable

Column Names:

+Q1.L_ORDERKEY+Q2.\$RID\$+Q2.\$P-ROWID\$

+Q2.0_ORDERPRIORITY+Q2.0_ORDERKEY

Output Streams:

7) To Operator #3

Estimated number of rows: 250002
Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:

+Q2.0_ORDERPRIORITY(A)+Q2.\$RID\$(A)
+Q2.\$P-ROWID\$(A)

5) MSJOIN: (Merge Scan Join)
Cumulative Total Cost: 222162
Cumulative CPU Cost: 3.27589e+09
Cumulative I/O Cost: 16810.7
Cumulative Re-Total Cost: 222162
Cumulative Re-CPU Cost: 3.27589e+09
Cumulative Re-I/O Cost: 16810.7
Cumulative First Row Cost: 165.969
Estimated Bufferpool Buffers: 16811.9
Remote communication cost:397356

Arguments:

EARLYOUT: (Early Out flag)
LEFT
INNERCOL: (Inner Order Columns)
1: Q1.L_ORDERKEY(A)
OUTERCOL: (Outer Order columns)
1: Q2.0_ORDERKEY(A)
TEMPSIZE: (Temporary Table Page Size)
4096

Predicates:

5) Predicate used in Join
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 3.99997e-06

Predicate Text:

(Q1.L_ORDERKEY = Q2.0_ORDERKEY)

Input Streams:

2) From Operator #6

Estimated number of rows: 250002
Number of columns: 4
Subquery predicate ID: Not Applicable

Column Names:

+Q2.0_ORDERKEY(A)+Q2.\$RID\$+Q2.\$P-ROWID\$
+Q2.0_ORDERPRIORITY

5) From Operator #9

Estimated number of rows: 1.99876
Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:

+Q1.L_ORDERKEY(A)+Q1.L_COMMITDATE(A)
+Q1.L_RECEIPTDATE(A)

Output Streams:

6) To Operator #4

Estimated number of rows: 250002
Number of columns: 5
Subquery predicate ID: Not Applicable

Column Names:

+Q1.L_ORDERKEY+Q2.\$RID\$+Q2.\$P-ROWID\$
+Q2.0_ORDERPRIORITY+Q2.0_ORDERKEY

6) SHIP : (Ship)

Cumulative Total Cost: 37714.3
Cumulative CPU Cost: 4.40279e+08
Cumulative I/O Cost: 9486.34
Cumulative Re-Total Cost: 168.177
Cumulative Re-CPU Cost: 3.56048e+08
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 100.032

Estimated Bufferpool Buffers: 9486.51
Remote communication cost:139105

Arguments:

CSERQY : (Remote common subexpression)

FALSE

DSTSEVER: (Destination (ship to) server)

- (NULL).

JN INPUT: (Join input leg)

OUTER

RMTQTX : (Remote statement)

SELECT AO."O_ORDERKEY", AO."O_ORDERPRIORITY", AO.ROWID FROM
"IITEST"."ORDERS" AO WHERE (1 <= AO."O_ORDERKEY") AND (AO."O_ORDERKEY" <= 1000000) ORDER BY 1 ASC

SRCSEVER: (Source (ship from) server)

ORASERV

STREAM : (Remote stream)

FALSE

Input Streams:

1) From Object ORA.ORDERS

Estimated number of rows: 1.5e+07

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$

Output Streams:

2) To Operator #5

Estimated number of rows: 250002

Number of columns: 4

Subquery predicate ID: Not Applicable

Column Names:

+Q2.O_ORDERKEY(A)+Q2.\$RID\$+Q2.\$P-ROWID\$

+Q2.O_ORDERPRIORITY

9) FILTER: (Filter)

Cumulative Total Cost: 184277

Cumulative CPU Cost: 2.47251e+09
Cumulative I/O Cost: 7324.35
Cumulative Re-Total Cost: 253.012
Cumulative Re-CPU Cost: 5.35653e+08
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 65.049
Estimated Bufferpool Buffers: 7325.35
Remote communication cost:258251

Arguments:

JN INPUT: (Join input leg)
INNER

Predicates:

5) Residual Predicate
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 3.99997e-06

Predicate Text:

(Q1.L_ORDERKEY = Q2.O_ORDERKEY)

Input Streams:

4) From Operator #10

Estimated number of rows: 499694
Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:

+Q1.L_ORDERKEY(A)+Q1.L_COMMITDATE(A)
+Q1.L_RECEIPTDATE(A)

Output Streams:

5) To Operator #5

Estimated number of rows: 1.99876
Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:

```

-----
+Q1.L_ORDERKEY(A)+Q1.L_COMMITDATE(A)
+Q1.L_RECEIPTDATE(A)

```

10) SHIP : (Ship)

```

Cumulative Total Cost: 184277
Cumulative CPU Cost: 2.47251e+09
Cumulative I/O Cost: 7324.35
Cumulative Re-Total Cost: 253.012
Cumulative Re-CPU Cost: 5.35653e+08
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 65.049
Estimated Bufferpool Buffers: 7325.35
Remote communication cost:258251

```

Arguments:

```

CSERQY : (Remote common subexpression)
FALSE
DSTSEVER: (Destination (ship to) server)
- (NULL).

```

RMTQTX : (Remote statement)

```

SELECT AO."L_ORDERKEY" FROM "TPCD"."LINEITEM" AO WHERE
(AO."L_COMMITDATE" < AO."L_RECEIPTDATE") AND (AO."L_ORDERKEY" <= 1000000) AND
(1 <= AO."L_ORDERKEY") ORDER BY 1 ASC, AO."L_COMMITDATE" ASC,
AO."L_RECEIPTDATE" ASC FOR READ ONLY

```

```

SRCSEVER: (Source (ship from) server)
DB2SERV
STREAM : (Remote stream)
FALSE

```

Input Streams:

3) From Object DB2.LINEITEM

```

Estimated number of rows: 5.99861e+07
Number of columns: 4
Subquery predicate ID: Not Applicable

```

Column Names:

```

+Q1.$RID$+Q1.L_RECEIPTDATE+Q1.L_COMMITDATE
+Q1.L_ORDERKEY

```

Output Streams:

4) To Operator #9

Estimated number of rows: 499694
Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:

+Q1.L_ORDERKEY(A)+Q1.L_COMMITDATE(A)
+Q1.L_RECEIPTDATE(A)

Objects Used in Access Plan:

Schema: DB2

Name: LINEITEM

Type: Nickname

Time of creation: 2004-06-11-21.33.19.482113

Last statistics update: 2004-06-11-22.38.00.532171

Number of columns: 16

Number of rows: 59986052

Width of rows: 133

Number of buffer pool pages: 2081039

Distinct row values: No

Tablespace name:

Tablespace overhead: 24.100000

Tablespace transfer rate: 0.900000

Source for statistics: Single Node

Prefetch page count: 32

Container extent page count: 32

Schema: ORA

Name: ORDERS

Type: Nickname

Time of creation: 2004-06-11-21.33.10.349783

Last statistics update: 2004-06-11-22.32.45.545670

Number of columns: 9

Number of rows: 15000000

Width of rows: 49

Number of buffer pool pages: 443840

Distinct row values: No

Tablespace name:

Tablespace overhead: 24.100000

Tablespace transfer rate: 0.900000

Source for statistics: Single Node

Prefetch page count: 32

Container extent page count: 32

The SQL text in the RMTQTEXT field of SHIP operators 6 and 10 can then be used to find matches in the dynamic cache, as highlighted in Example 4-8 on page 136.

The key fields of interest in the dynamic cache output of Example 4-8 on page 136 for the user-entered query and the two remote SQL fragments are as follows:

1. **Number of executions**, which is 374 for the user-entered query as well as the two remote SQL fragments.
2. **Total execution time (sec.ms)**, which is 16984.947355 for the user-entered query, and 13144.507706 and 2724.553063 for the two remote SQL fragments respectively.
3. **Total user cpu time (sec.ms)** and **Total system cpu time (sec.ms)**, which is 2009.080000 and 85.490000 for the user entered query, while it is zero for both the remote SQL fragments.
4. **Rows read** is 3578 for the user-entered query, which is the number of rows returned to the user; while 236474590 and 93500000 respectively for each of the remote SQL fragments, which indicates the number of rows returned to the federated source from each corresponding data source respectively.
5. Other fields of interest include **Statement sorts**, **Statement sort overflows**, and **Total sort time**, which only apply to the user-entered query, and have values 374, zero, and 322696 respectively.

Note: To obtain the average elapsed and CPU times, as well as the number of rows returned, you must divide the values shown by the **Number of executions**.

We can derive the following information from these metrics:

- ▶ The average number of rows returned from each data source to the federated server is $(236474590 / 374) = 632285$ rows and $(93500000 / 374) = 250000$ rows, respectively.
- ▶ The average elapsed time for the user query is $(16984.94 / 374) = 45.41$ seconds, while that of each remote SQL fragment is $(13144.50 / 374) = 35.15$ seconds and $(2724.55 / 374) = 7.28$ seconds.

Since the access to the remote data sources occur serially, the total elapsed time at the remote data sources is $(35.15 + 7.28) = 42.43$ seconds.

Attention: In our example, given that the total query elapsed time is 45.41 seconds, it is clear that the predominant elapsed time of the query is spent at the remote data sources. $((42.43 / 45.41) \times 100) = 93.44\%$ of the query elapsed time is accounted for at the remote data sources, indicating them to be the potential source of the performance problem. In fact, the major portion of the elapsed time is at the remote SQL fragment accessing the DB2 LINEITEM table.

4.2.4 Federated server related

As mentioned earlier, a number of potential causes of poor performance need to be investigated when the problem appears to be federated server related. While there is no hard and fast rule about the sequence in which the following items need to be investigated, current experience suggests the sequence documented to be most effective.

1. Statistics
2. Index information
3. Pushdown
4. Joins
5. Parallelism

Each of these items is discussed in detail in the following subsections.

Note: This sequence may not be the most effective sequence in your environment, and you are encouraged to adopt one based on personal experience.

Statistics

Current and accurate statistics about nickname objects are critical to the DB2 optimizer choosing an optimal access path for a federated query. Statistics are collected at nickname creation, and when the **NNSTAT** stored procedure or Statistics Update is invoked from the DB2 Control Center, as shown in Figure 4-4 on page 153.

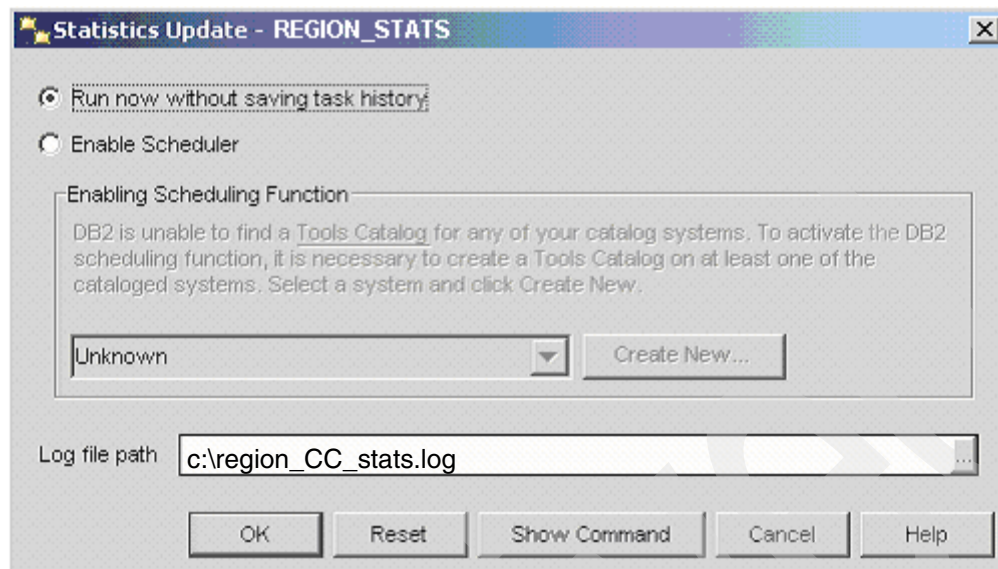


Figure 4-4 Statistics Update in DB2 Control Center

In Figure 4-4 the federated server retrieves the statistics for the nickname REGION_STATS, and writes the log to the **region_cc_stats.log** file.

Attention: These statistics are retrieved from the remote data source catalog, and do not necessarily reflect the actual data in a table. For example, the REGION_STATS table at the remote data source may actually contain 10 million rows, while the remote data source catalog shows 5 million rows since statistics had not been gathered of late on the REGION_STATS table at the remote data source. The Statistics Update facility will reflect 5 million rows at the federated source, and not the 10 million rows that actually exist in the remote table.

Also, column value distribution statistics are important for helping the optimizer estimate the cost of alternative access plans and picking the access plan that has the best chance of fastest execution times. The optimizer estimates the number of rows that will result from joins and filters using both the CARD value in SYSCAT.TABLES for the nicknames involved and also the value distribution statistics for the nickname columns involved in SYSCAT.COLUMNS and SYSCAT.INDEXES. COLCARD values for the relevant nickname columns in SYSSTAT.COLUMNS help the optimizer to estimate the cost more precisely than if just the CARD value was available in SYSSTAT.TABLES. HIGH2KEY/LOW2KEY values in SYSSTAT.COLUMNS and FULLKEYCARD/FIRSTKEYCARD in SYSSTAT.INDEXES help the optimizer be more precise in its cost estimates.

To determine whether there is a possible discrepancy in the statistics between the federated server and the remote data source, the contents of the dynamic cache and the corresponding EXPLAIN output may provide some clue.

In Example 4-8 on page 136, the **(Rows read / Number of executions)** computation field shows a particular query returning an average of 632285 rows from the DB2 LINEITEM table, while an average of 250000 rows are returned from the Oracle ORDERS table.

The Access plan graph in the **db2exfmt** output for the same query (Example 4-9 on page 138) shows the nickname ORDERS (SHIP operator 6) having a cardinality of 1.5e+07 rows, while the nickname LINEITEM (SHIP operator 10) has a cardinality of 5.99861e+07 rows. Additionally, the SHIP operator 6 shows an estimate of 250002 rows being returned to the federated server from the Oracle data source, while 499694 rows are estimated to be returned from the DB2 data source according to the SHIP operator 10.

The discrepancy between the actual rows and the estimated rows returned from the Oracle data source is $(250000 - 250002) = 2$ rows, which is pretty accurate. The discrepancy between the actual rows and the estimated rows returned from the DB2 data source is $(632285 - 499694) = 133691$ rows, which may be significant. This discrepancy between the optimizer's estimate of number of rows

returned and the actual number of rows that are returned may be attributed to the following causes:

- ▶ The nickname statistics, including the column value distribution statistics, are not accurate. This results in the optimizer estimating the size of intermediate result sets and size of the final results sets on inaccurate information, before even accounting for anomalies in the data as described next.
- ▶ The actual values in the columns of the data are not evenly distributed across all the rows. The optimizer assumes uniform distribution of values in the column statistics for the nickname across all the records. The classic example of this is the product key values for milk products in a supermarket data warehouse. The different product key values for milk products are a small percentage of the total number of different key values in the transaction tables, but they constitute a disproportionately high percentage of the transaction records since most people buy milk when they go to the market.

A significant difference is reason to consider running the stored procedure **NNSTAT** or the Statistics Update facility on the nickname corresponding to the DB2 data source, which in this case is the LINEITEM nickname.

Note: As mentioned earlier, since the Statistics Update facility or **NNSTAT** stored procedure only copies the remote data source catalog contents onto the nickname statistics, there is no guarantee that the nickname statistics will be accurate.

It is strongly recommended that the remote data source statistics be updated first before executing the Statistics Update facility or the **NNSTAT** stored procedure.

“Nickname statistics and index information” on page 69 describes the considerations when using the Statistics Update facility or the **NNSTAT** stored procedure.

Index information

Having index information on nicknames is essential to providing the DB2 optimizer with additional information in order for the DB2 optimizer to choose an optimal access path for a federated query.

As in the case of statistics, index information on nicknames is also collected at nickname creation.

The updateable view SYSSTAT.INDEXES can be used to add FULLKEYCARD and FIRSTKEYCARD after an INDEX...SPECIFICATION ONLY record has been

added to the catalog to provide index/column distribution statistics for the optimizer.

“Nickname statistics and index information” on page 69 describes the considerations when synchronizing index information using the Statistics Update facility or the **NNSTAT** stored procedure.

Pushdown

It is generally desirable for predicates to be pushed down to the remote data source as much as possible to reduce the number of rows returned to the federated server for additional processing before returning results back to the user.

Some of the predicates cannot be pushed down to the remote data source and have to be processed at the federated server for reasons such as the unavailability of the required functionality at the remote data source, and the setting of the DB2 II server option **PUSHDOWN** = 'N'. Other predicates are candidates for being pushed down, but may not be, depending upon DB2 II server options such as **DB2_MAXIMAL_PUSHDOWN** and DB2 optimizer cost evaluations. Refer to “Pushdown” on page 78 for a detailed discussion of the pushdown concept.

Identifying non-pushdownable predicates

Assuming that the DB2 II server option **PUSHDOWN** is set to 'Y' (default), predicates that *cannot* be pushed down can be identified by setting the **DB2_MAXIMAL_PUSHDOWN** server option to 'Y', and viewing all operators above the SHIP operator(s) in **db2exfmt** output as being non-pushdownable, as shown in Example B-13 on page 586.

The non-pushdownable predicates in Example B-13 on page 586 are five SHIP operators (7, 18, 25, 29, and 31) in the **db2exfmt** output, and all operators above them are considered non-pushdownable operators such as NLJOIN (2, 5, and 6), MSJOIN (15), TBSCAN (3, 16 and 23), SORT (4, 17 and 24), GRPBY (14), and FILTER (13 and 22).

Identifying pushdownable predicates that were not pushed down

Assuming that the DB2 II server option **PUSHDOWN** is set to 'Y' (default), predicates that are pushdownable but were not for a given query can be identified by comparing operators above the SHIP operators in the **db2exfmt** output of **DB2_MAXIMAL_PUSHDOWN** = 'Y', as shown in Example B-13 on page 586, with the **db2exfmt** output of **DB2_MAXIMAL_PUSHDOWN** = 'N', as shown in Example B-12 on page 554.

The predicates that were not pushed down in Example B-12 on page 554 are six SHIP operators (8, 17, 24, 28, 31 and 33) in the **db2exfmt** output, and all

operators above them were either non-pushdownable or pushdownable but not pushed down for cost reasons. The list of operators not pushed down are NLJOIN (2, 5, 6, and 7), MSJOIN (14), TBSCAN (3, 15, and 22), SORT (4, 16, and 23), GRPBY (13), and FILTER (12 and 21).

Comparing the **db2exfmt** outputs in Example B-13 on page 586 and Example B-12 on page 554, we can identify that the difference is in the number of SHIP operators and NLJOIN operators.

- ▶ Example B-13 on page 586 with the DB2_MAXIMAL_PUSHDOWN = 'Y' option has one fewer SHIP operator since it pushes a 3-way join of the nicknames SUPPLIER, PART, and PARTSUPP via SHIP operator 7, thereby resulting in one fewer nested loop join.

With DB2_MAXIMAL_PUSHDOWN 'Y' we can see which joins and filters DB2 II's PDA determines cannot be pushed down. If we compare the SQL statement with the access plan we see that:

- Though PARTSUPP and SUPPLIER are referenced in both the main select at the top of the statement and in the sub-select near the bottom of the statement, DB2 II cannot send just one SQL statement to the data source 'DB2' for both of these select clauses. It is evident that the minimal number of SHIPs to data source 'DB2' is two—one for the join of PART, PARTSUPP, and SUPPLIER in the main select, and another for the join of PARTSUPP and SUPPLIER in the sub-select.
 - The reason for this restriction is that SUPPLIER is joined to a NATION at 'ORA' in both the main select and the sub-select.
 - For a similar possible reason, DB2 II cannot perform just one access to 'ORA' to get data from NATION and REGION for use in the main select and the sub-select.
- ▶ Example B-12 on page 554 with the DB2_MAXIMAL_PUSHDOWN = 'N' option has an additional SHIP operator since it chooses to join nicknames PART and PARTSUPP via the SHIP operator 8, the result of which is then joined (NLJOIN operator 7) with the results of the FILTER operator 12. The results of NLJOIN operator 7 is then joined with the results of SHIP operator 28 that accesses SUPPLIER.

Note: An alternative approach to identifying predicates not pushed down is to look at the SQL in the RMTQTX field of the SHIP operator in conjunction with the SQL in the Optimized Statement section of **db2exfmt** output. Bear in mind that the Optimized Statement does not always identify the SQL statement used in access path selection.

The Total cost of 696494 timerons for Example B-12 on page 554 with DB2_MAXIMAL_PUSHDOWN = 'N' is estimated by the DB2 optimizer to be

lower than the Total cost of 712543 timerons for Example B-13 on page 586 with DB2-MAXIMAL_PUSHDOWN = 'Y'. This clearly demonstrates the functionality of the DB2_MAXIMAL_PUSHDOWN server option and the reason why the default setting is 'N'.

Attention: In the final analysis, what really matters is not what the DB2 optimizer estimates to be the optimal access path based on timerons, but the actual run times experienced by the user. This requires accurate measurements in a representative runtime environment.

Joins

DB2 supports three potential join strategies: nested loop, merge scan and hash joins. Each of these join methods is optimal under certain conditions.

Table 4-2 provides a very high-level overview of conditions under which one of the three join strategies is favored over the other. It is not a comprehensive list. Refer to the *IBM DB2 UDB Administration Guide: Performance*, SC09-4821-00, for a detailed discussion of these join methods and join strategies.

Table 4-2 Join strategies

Criteria	Nested loop join	Merge scan or Hash join
Only limited memory available	Better	Worse, since memory is needed for sorting or building the hash table.
Need the first row quickly	Better	Worse, since a sort or hash is needed before a row can be returned.
Selecting only a few rows from the inner table	Better	Worse.
Parallelism	Best	Okay.
Index available on inner table	Best	Okay.
No equality predicates	Okay	Requires at least one equality predicate. For hash join, the data type, scale and precision must match perfectly.
Very large numbers of qualifying rows for each input to the join	Worse	Not much better. Hash and sort will almost have equal costs.

Criteria	Nested loop join	Merge scan or Hash join
One very small input and one very large input to the join	Worse	Hash join better than merge.

When a federated query joins nicknames from different data sources, the join occurs at the federated server and can consume sort heap and buffer pool resources depending upon the join strategy adopted by the DB2 optimizer. Both the hash join (HSJOIN) and merge scan join (MSJOIN) techniques use memory. In the case of the hash join, DB2 II creates a hash table using an allocation from SORTHEAP. Merge scan also uses SORTHEAP allocation.

db2exfmt output identifies the type of join chosen (NLJOIN, MSJOIN and HSJOIN operators), the predicates involved, and estimates the number of rows in the input to the join. Sometimes, if the join input is from SHIP operators, the content of the dynamic cache (**Rows read** field) can provide actual number of rows input to the join rather than optimizer estimates. This information is critical to evaluating whether or not the choice of a particular join method may be the most optimal for a given query.

In Example B-12 on page 554, the nested loop join (NLJOIN operator 7) estimates 36042.4 rows from the SHIP operator 8 (outer table) being joined with an estimated 1.16257e-05 rows returned from the FILTER operator 12 (inner table). The predicates involved in the join are identified in the Predicate Text of the NLJOIN operator 7 as being Q9.PS_SUPPLYCOST = Q6.\$C0.

This information about the number of rows input to the join, the predicates involved, and configuration information (sort heap and buffer pool) can be used in conjunction with the join strategies information shown in Table 4-2 on page 158 to assess whether the join method chosen by the DB2 optimizer is appropriate for the given query.

Here again, in the final analysis, what really matters is not what the DB2 optimizer estimates to be the optimal access path based on timerons, but the actual run times experienced by the user. This requires accurate measurements in a representative runtime environment.

Attention: For federated queries that join nicknames from different data sources, the likelihood of data type mismatches between the joined columns is higher because of the default data type mapping that occurs when the nicknames are defined. Such a mismatch would inhibit the choice of the hash join method on that predicate, which would otherwise have been the optimal join method for the given query.

Another consideration is that the quality of the statistics gathered for a nickname may result in less accurate estimation of rows returned from a data source (for example, when distribution statistics are not available for a particular data source), thereby causing a less optimal join method to be chosen.

Parallelism

Intra-partition and inter-partition parallelism are supported by DB2 II, as described in “DB2 server parallelism” on page 74 and “DB2 II wrapper DB2_FENCED option” on page 62.

The following settings influence the DB2 optimizer in considering the choice of intra-partition and inter-partition parallelism:

- ▶ Database manager configuration parameter INTRA_PARALLEL, which must be set to YES (default is NO) for intra-partition parallelism.
- ▶ For intra-partition parallelism, the database configuration parameter DFT_DEGREE must be set to ANY(-1) or > 1 (default is 1). This sets the default value of the CURRENT DEGREE special register and the DEGREE bind option.
- ▶ Database partition feature (DPF) for inter-partition parallelism is installed and configured.
- ▶ DB2 II wrapper option DB2_FENCED = 'Y' (default is 'N') for inter-partition parallelism.
- ▶ Computation partition group (CPG) for inter-partition parallelism even when no local data is involved in the federated query.

db2exfmt output provides information about the enabling of parallelism (**Parallelism** field in the Database Context section), while the Access Plan section indicates through the presence of operators BTQ, DTQ, LTQ, LMTQ, MBTQ and MDTQ whether parallelism (intra and/or inter) was chosen for portions of the access plan. These operators are briefly described in Table B-3 on page 446.

Example B-9 on page 490 shows **db2exfmt** output for a query with intra-partition parallelism enabled as indicated by the value *Intra-Partition Parallelism* in the

Parallelism field of the Database Context section. The access plan graph shows the presence of the LMTQ operator 4, which indicates intra-partition parallelism being chosen.

Example B-11 on page 534 shows **db2exfmt** output for a query with inter-partition parallelism enabled as indicated by the value *Inter-Partition Parallelism* in the Parallelism field of the Database Context section. The access plan graph shows the presence of the DTQ operator 2 and the BTQ operator 4, which indicates inter-partition parallelism being chosen. Intra-partition parallelism had *not* been enabled in Example B-11 on page 534.

Note: Intra-partition parallelism should be considered in federated environments only when local DB2 data is present in the federated database and when there are sufficient spare CPU cycles to cope with the increased CPU utilization. Intra-partition parallelism can occur with nicknames defined using both the fenced and trusted wrappers. A federated server should only be placed on a partitioned database server when the existing DB2 database is already partitioned. One should not consider partitioning a federated server that does not contain local DB2 data of sufficient volumes to warrant partitioning. For nicknames to take advantage of inter-partition parallelism, they must be defined using the fenced wrapper.

The Parallelism field in the Database Context section of the **db2exfmt** output will indicate the type of parallelism enabled, if any; while the presence/absence of table queue operators in the access plan graph of the **db2exfmt** output will indicate whether the DB2 optimizer chose parallelism for the access plan. **db2exfmt** output also provides estimates of rows returned by each operator. Dynamic cache metrics (**Rows read** field), if appropriate, provide the actual number of rows returned, which is a more refined approach to assess whether parallelism should be enabled for the given query.

Here again, in the final analysis, what really matters is not what the DB2 optimizer estimates to be the optimal access path based on timerons, but the actual run times experienced by the user. This requires accurate measurements in a representative runtime environment.

4.2.5 Remote data source related

The remote data sources may be autonomously managed environments, in which case it is up to the remote administrator to apply appropriate tuning strategies after being given details of the remote SQL fragment such as the text of the SQL statement, elapsed time, and number of rows returned to the federated server as obtained from the dynamic cache.

When the remote data source is a relational database, we need to investigate the number of rows returned to the federated server, the degree of pushdown of the predicates in the user-entered query to determine if any of the DB2 II configuration options (related to DB2 database manager, DB2 II server, or DB2 II nickname columns) need review, or whether the query syntax itself is inhibiting pushdown. The same procedures as discussed in 4.2.4, “Federated server related” on page 152, apply here. A performance problem can be isolated to a particular data source by either reviewing snapshot monitor information, or by natively executing the SQL fragment, which is pushed down to the remote data source using the data sources client. If a problem is suspected at the remote data source, then the federated server DBA and the DBA for the remote data source should work together to isolate the cause of the problem.

4.3 Monitoring best practices

DB2 provides a number of monitoring capabilities for problem diagnosis as well as tuning the performance of the system.

The information collected by the snapshot monitor¹ of the Database System Monitor in particular is controlled by a set of monitor switches defined in the database manager configuration file. these switches and their default settings are described in Table 4-3.

Attention: There is a considerable amount of basic monitoring data that is not under monitor switch control, and will always be collected regardless of switch settings. Figure 4-5 shows the monitoring data collected in relation to monitor switch settings.

Table 4-3 Snapshot monitor switches

Monitor switch	Default	Description
DFT_MON_BUFPOOL	OFF	Buffer pool activity information such as number of reads and writes, and time taken
DFT_MON_LOCK	OFF	Lock wait times and deadlock-related information
DFT_MON_SORT	OFF	Sorting information such as number of heaps used and sort performance

¹ The snapshot monitor is used to capture information about the database and any connected applications at a specific time. Snapshots are useful for determining the status of a database system. Taken at regular intervals, they are also useful for observing trends and foreseeing potential problems.

Monitor switch	Default	Description
DFT_MON_STMT	OFF	SQL statement information such as start/stop time, and statement identification
DFT_MON_TABLE	OFF	Table activity information such as rows read and written
DFT_MON_TIMESTAMP	ON	Times and timestamp information
DFT_MON_UOW	OFF	Unit of work information such as start/end times and completion status

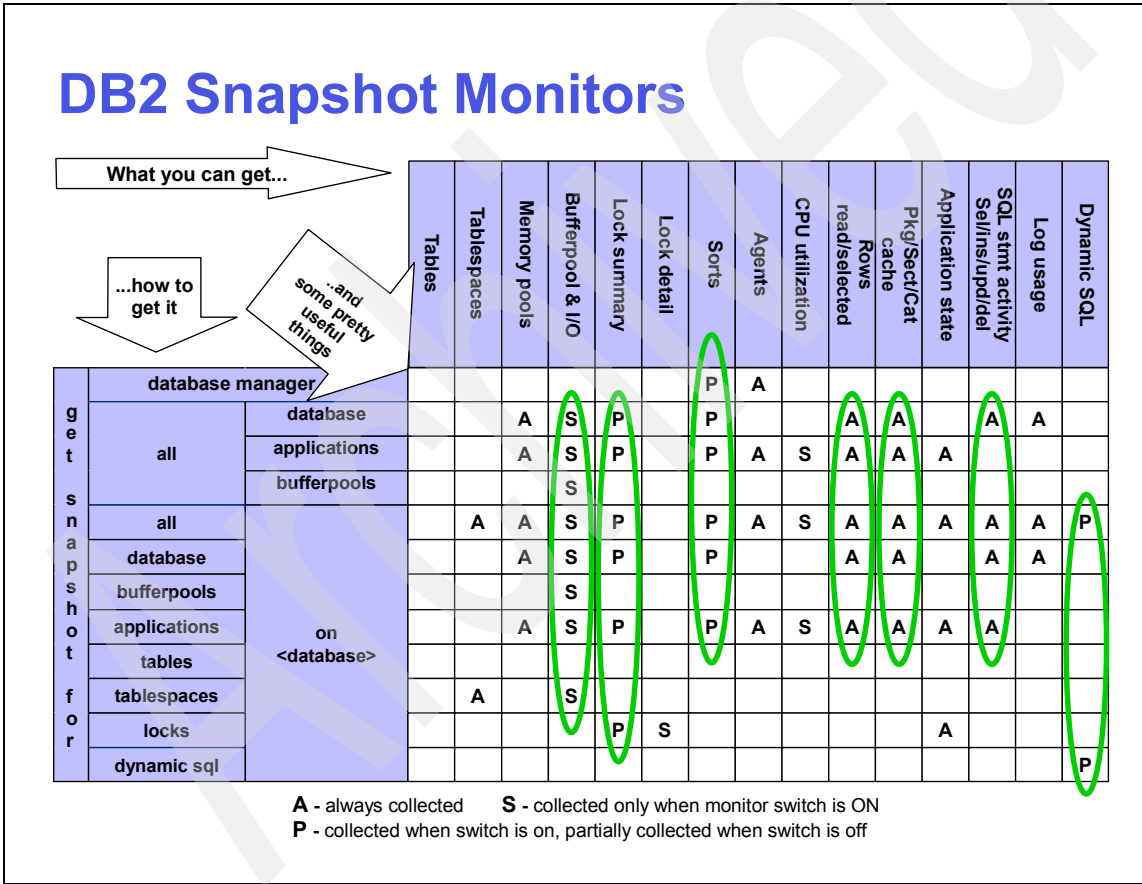


Figure 4-5 DB2 snapshot monitor syntax and data collection

Note: Event Monitors^a are not affected by monitor switches in the same way as snapshot monitoring applications. When an event monitor switch is defined, it automatically turns on the instance-level monitor switches required by the specific event types.

a. Event monitor differs from the snapshot monitor in that it is used to collect information about the database and any connected applications when specified events occur. Events represent transitions in database activity, for instance, connections, deadlocks, statements, and transactions. Event monitor does not provide added value for federated queries.

Refer to the *IBM DB2 UDB System Monitor Guide and Reference*, SC09-4847, for more information on the monitor switches, including setting them, taking snapshots, and other general considerations.

4.3.1 Performance considerations

There are overheads associated with collecting the Database System Monitor. This includes collecting the data when the monitor switches are set, as well as the processing cost of frequently retrieving this information via the **get snapshot** and **flush event monitor** commands.

Each monitor switch setting imposes a certain overhead that is dependent upon the nature of the workload. The overheads are predominantly related to CPU consumption rather than waits due to concurrency issues.

Typical overheads are as follows:

- ▶ All switches set: Approximately 5–10 percent, but will vary depending upon the workload.
- ▶ The DFT_MON_LOCK setting imposes an overhead of between 1–3 percent depending upon the frequency of snapshot requests.
- ▶ The DFT_MON_STMT setting with dynamic SQL workloads imposes an overhead of between 5–10 percent proportional to statement throughput. The timestamp switch setting needs to be considered in the overhead experienced.
- ▶ The DFT_MON_TIMESTAMP setting (default is ON) overhead can become significant when CPU utilization approaches 100 percent. When this setting is ON, the database manager issues timestamp operating system calls when determining time or timestamp-related monitor elements. When this setting is OFF, the overhead of other switch settings is greatly reduced.

Important: Since the overhead associated with each monitor switch setting is completely dependent upon the workload and the frequency of snapshot requests, you should determine the overheads in your specific environment through careful measurements. The overheads listed earlier are merely provided as guidelines to be used prior to detailed measurement in your environment.

4.3.2 Best practices

We recommend the following best practices for the different types of monitoring.

Routine monitoring

As discussed in 2.3.1, “Routine monitoring” on page 35, the objectives of this type of monitoring are to collect information about the workload and stress on the system during periods of normal and peak periods for capacity planning purposes, as well as identifying potential performance problems in the future.

Note: Routine monitoring overhead should typically not exceed 5 percent.

From a problem determination point of view, the objective is to detect deteriorating trends of key performance drives, and then perform exception monitoring to pinpoint the problem and resolve it before it gets out of hand.

Attention: Routine monitoring requires a history repository to determine trends, and reporting mechanisms to alert potential problem conditions using thresholds and alerts.

We did not have such a history repository available, and just asserted the existence of a particular trend for our problem diagnosis purposes.

We recommend the following monitor switch settings, database configuration settings, and **get snapshot** command frequency for routine monitoring of a federated database environment:

- ▶ Set all switches to ON except DFT_MON_STMT and DFT_MON_LOCK.
- ▶ Federated systems are similar to business intelligence environments (largely read only), and therefore the frequency of snapshot requests should be low for lightly loaded systems (say every 300 seconds), and higher for highly dynamic systems (say every 60 seconds).
- ▶ Let DIAGLEVEL² default to 3.

² Specifies the type of diagnostic errors recorded in the db2diag.log.

- ▶ Let the NOTIFYLEVEL³ default to 3.
- ▶ Let HEALTH_MON⁴ default to ON.

Attention: These settings should be evaluated in the context of overheads incurred in your specific environment.

Online/event monitoring

As discussed in 2.3.2, “Online/realtime event monitoring” on page 36, the objective of this type of monitoring is to be on the lookout for specific events that may either identify a specific problem, or portend problems in the near to immediate future, in order to take prompt corrective action. Near to immediate future implies minutes rather than hours.

The key to this type of monitoring is that it involves looking for specific events in a short interval of time (short history) that are known to degrade performance, and having the option to take prompt corrective action to rectify the problem. In other words, there probably needs to be a very short delay between information collection and a corrective response. One example of such an event is the invalidation of a nickname, which needs to be addressed promptly to ensure that business objectives are not being compromised.

Note: The need to minimize the overhead of online/realtime monitoring is critical given that most problems manifest themselves at peak loads.

Note that online/realtime monitoring is similar to routine monitoring, since its alerts will enable you to bypass certain validations in the DB2 hypotheses hierarchy by virtue of the fact that these alerts point to a specific problem area.

The Health Monitor and Health Center are the primary tools for online/realtime monitoring in DB2.

Exception monitoring

As described in 2.3.3, “Exception monitoring” on page 37, this type of monitoring is required when you discover or suspect a problem, and need to identify its root cause in order to apply the appropriate corrective action to fix the problem.

Unlike routine and event monitoring, which are planned occurrences and are designed to have low overheads on the managed system, exception monitoring

³ Specifies the type of administration notification messages that are written to the administration notification log.

⁴ Specifies whether the DB2 instance, its associated databases and database objects should be monitored using the Health Center's health indicators.

is driven by problem situations and may impose significant overheads on the managed system.

4.4 Problem scenarios

In the following sections, we document the procedures we followed for identifying the following frequently encountered performance problems in a DB2 II environment using both routine monitoring and exception monitoring:

- ▶ Missing or incorrect statistics/index information
- ▶ Poorly tuned sort heap and buffer pools
- ▶ Missing or unavailable MQTs
- ▶ Incompatible data types on join columns
- ▶ Pushdown problems
- ▶ Default DB2_FENCED wrapper option with DPF

Attention: Two important points need to be noted about these scenarios, as follows:

- ▶ The workload and environments were artificially contrived to produce the relevant problem condition for the problem diagnosis exercise; therefore, certain settings can clearly be seen to be “inappropriate” in real-world environments.
- ▶ The emphasis of these scenarios is on problem diagnosis, and *not* on problem resolution per se. Best practices for problem resolution are discussed briefly, but not applied to demonstrate the elimination of the problem.

We have organized the discussion of each scenario as follows:

- ▶ Triggering event
- ▶ Hypotheses and their validation
- ▶ Root cause of the problem
- ▶ Apply best practices

We used a common test environment for these scenarios, as described in 4.4.1, “Federated test environment” on page 167.

4.4.1 Federated test environment

Our pseudo production environment for our problem scenarios is shown in Figure 4-6 on page 168.

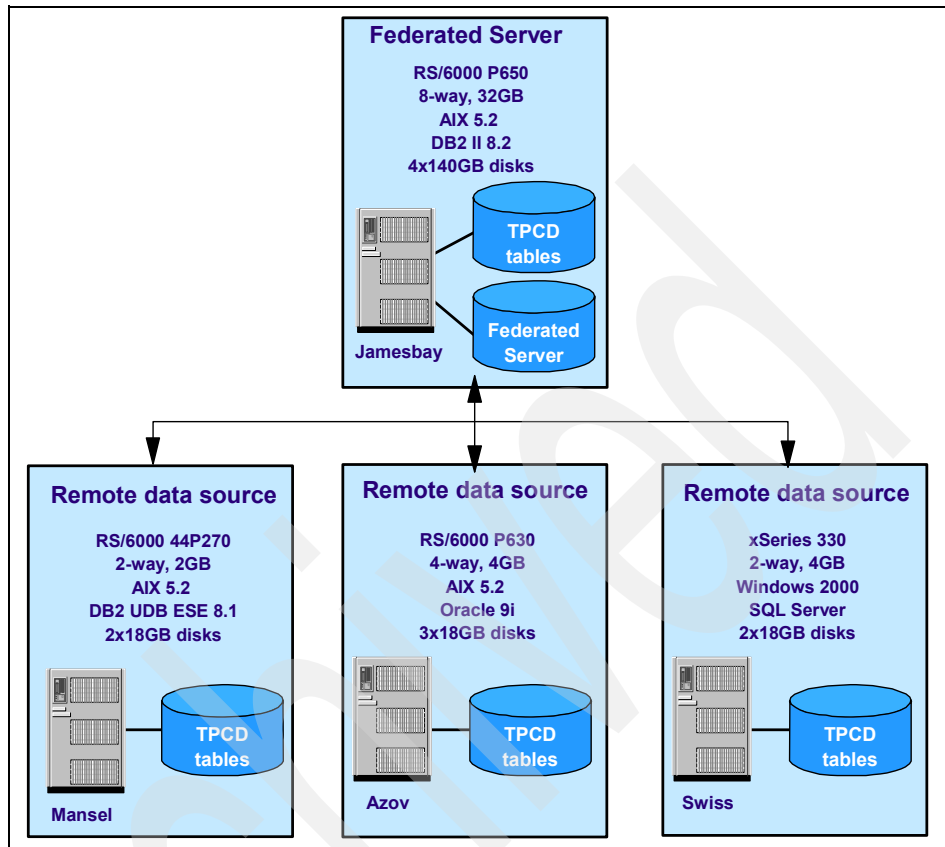


Figure 4-6 Federated test environment

The environment consists of:

- ▶ DB2 II V8.2 federated server instance on an 8-way 32 GB P650 (Jamesbay) running AIX 5.2
- ▶ Oracle 9i database server on a 4-way 4 GB P630 (Azov) running AIX 5.2
- ▶ DB2 ESE V8.2 database server on 2-way 2 GB 44P 270 (Mansel) AIX 5.2
- ▶ Microsoft SQL Server 2000 server on a 2-way 4 GB xSeries® 330 (Swiss) running Windows 2000

Each of the database servers contains identical tables and table names corresponding to the TPCD benchmark, but were assigned different schema names as follows:

- ▶ DB2 UDB ESE used the schema DB2.
- ▶ Oracle used the schema ORA.

- SQL Server 2000 used the schema MSS.
- Local tables on DB2 used the schema TPCD. These tables were located in the same database as the federated server.

Each database contains the eight TPCD tables shown in Figure 4-7, and all have the same cardinality listed in Table 4-4.

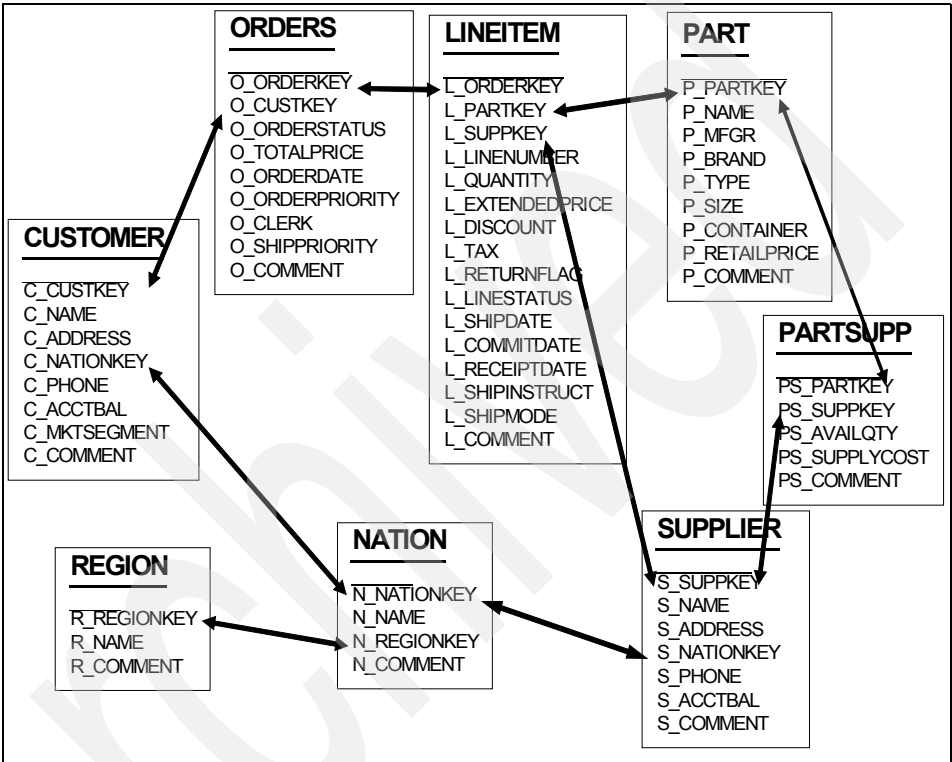


Figure 4-7 TPCD tables

Table 4-4 TPCD tables cardinality

Table name	Relationship between records in different tables	Number of rows
ORDERS	One record per order	1,500,000 orders
LINEITEMS	One record per order item	6,000,000 line items
PART	One record per part	200,000 parts
SUPPLIER	One record per supplier	10,000 suppliers

Table name	Relationship between records in different tables	Number of rows
PARTSUPP	One record for each supplier for a part	
CUSTOMER	One record per customer	200,000 customers
NATION	One record per nation	25 nations
REGION	One record per region	5 regions

Nicknames were created at the DB2 II federated server (Jamesbay) for all of the tables, using the appropriate schema name to identify the intended server. For example, the nickname MSS.NATION refers to the NATION table on the SQL Server 2000 database server. This naming convention provided us with the flexibility to vary the location of the nickname data for our different scenarios without having to change the SQL statements.

Note: We had multiple instances on Jamesbay, both 32-bit and 64-bit instances, which we used as the scenario demanded.

We used the twenty-two TPCD benchmark queries as well as additional home-grown queries as our application workload. For details about the TPCD benchmark and its queries, please refer to the Transaction Processing Performance Council Web site:

<http://www.tpc.org/>

Note: We simulated concurrent user access to our federated test environment using local home-grown driver scripts and local connections supplying input via the command line. We did *not* employ a Web application server.

We had the following monitor-level settings for this environment:

- ▶ All switches to ON except DFT_MON_LOCK.
- ▶ Frequency of snapshot requests was on demand and varied by scenario
- ▶ Let DIAGLEVEL default to 3.
- ▶ Let the NOTIFYLEVEL default to 3.
- ▶ Let HEALTH_MON default to ON.

4.4.2 Missing or incorrect statistics/index information

As mentioned earlier, statistics and index information is automatically collected when a nickname is created. However, it is the DBA's responsibility to ensure that

this information is kept current as changes occur in objects at the remote data source. The statistics and index information for a nickname can lose currency causing the DB2 optimizer to make suboptimal access path decisions for queries accessing these nicknames.

In this scenario, we show how incorrect statistics for a nickname may be the cause of a performance problem.

Triggering event

Several user complained about poor response times for their queries. Some users said that their current queries were experiencing poor response times—the performance problem was in progress.

Hypotheses and validations

Since there were several user complaints about poor performance, as per Figure 4-3 on page 122, we decided to enter the DB2 hypotheses hierarchy shown in Figure 4-1 on page 117 and described in Example 4.2 on page 119, at the top of the hierarchy, and validated each hypothesis (excluding the Web application server since it was not applicable for our environment) in turn, as follows:

- ▶ Hypothesis 1: Network performance
- ▶ Hypothesis 2: Federated server performance
- ▶ Hypothesis 3: Federated database server performance
- ▶ Hypothesis 4: Federated application/query performance

Hypothesis 1: Network performance

We consulted the network administrator about the performance of the network and were informed that network performance was within normal bounds. Since both the federated server and the remote data source servers resided on the same network in our environment, we only needed to check the availability and responsiveness of one network.

Our own attempts to **ping** the federated server (jamesbay.almaden.ibm.com®) and the Oracle data source server (azov.almaden.ibm.com) returned very low round-trip times, as shown in Figure 4-8 on page 172.

```
C:\>Command Prompt

C:\>ping jamesbay.almaden.ibm.com

Pinging jamesbay.almaden.ibm.com [9.1.39.79] with 32 bytes of data:

Reply from 9.1.39.79: bytes=32 time<10ms TTL=254
Reply from 9.1.39.79: bytes=32 time<10ms TTL=254
Reply from 9.1.39.79: bytes=32 time<10ms TTL=254
Reply from 9.1.39.79: bytes=32 time<10ms TTL=254

Ping statistics for 9.1.39.79:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\>ping azov.almaden.ibm.com

Pinging azov.almaden.ibm.com [9.1.39.89] with 32 bytes of data:

Reply from 9.1.39.89: bytes=32 time<10ms TTL=254
Reply from 9.1.39.89: bytes=32 time<10ms TTL=254
Reply from 9.1.39.89: bytes=32 time<10ms TTL=254
Reply from 9.1.39.89: bytes=32 time<10ms TTL=254

Ping statistics for 9.1.39.89:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms

C:\>
```

Figure 4-8 ping DB2 II federated server and Oracle data source server (azov)

Network performance was very good. We therefore concluded that network performance was not the cause of the slow response times being experienced by the user.

Hypothesis 2: Federated server performance

We asked the system administrator of the federated server machine to verify that this machine was performing acceptably for the workload. Several utilities were used to review the system-level performance.

The `vmstat` command was used to show CPU utilization information on the federated server, as shown in Figure 4-9 on page 173.

```

jamesbay.almaden.ibm.com - PuTTY
$ vmstat 2 5
kthr      memory          page        faults        cpu
-----
 r  b   avm    fre  re   pi  po  fr   sr  cy   in   sy  cs  us  sy  id  wa
2  3 1784794 4201385    0    0    0   11   22    0 1752 7341 2859  3   1 95   2
0  0 1784798 4201381    0    0    0    0    0    0 1335 2818 2155  0   0 99   0
0  0 1784798 4201381    0    0    0    0    0    0 1316 2743 2102  0   0 99   0
0  0 1785965 4200609    0    0    0    0    0    0 1323 5665 2131  0   1 99   0
0  0 1784815 4201364    0    0    0    0    0    0 1304 4685 2089  0   1 99   0
$

```

Figure 4-9 *vmstat* command output from the federated server

The **iostat** command was used to show disk I/O utilization information on the federated server, as shown in Figure 4-10 on page 174.

```
jamesbay.almaden.ibm.com - PuTTY
$ iostat 2 2

tty:      tin      tout    avg-cpu:  % user   % sys    % idle   % iowait
          1.2      759.2          2.6     0.7     94.8     1.9

Disks:      % tm_act    Kbps      tps      Kb_read  Kb_wrtn
hdisk2      0.0        1.0        0.2     391044   268052
hdisk0      0.3        3.5        0.6     13371   2358435
hdisk3      0.0        1.0        0.2     384888   293580
hdisk1      0.0        1.0        0.2     403864   239544
cd0         0.0        0.0        0.0        0        0

tty:      tin      tout    avg-cpu:  % user   % sys    % idle   % iowait
          0.5      357.0          0.0     0.2     99.8     0.0

Disks:      % tm_act    Kbps      tps      Kb_read  Kb_wrtn
hdisk2      0.0        0.0        0.0        0        0
hdisk0      0.0        0.0        0.0        0        0
hdisk3      0.0        0.0        0.0        0        0
hdisk1      0.0        0.0        0.0        0        0
cd0         0.0        0.0        0.0        0        0
$
```

Figure 4-10 iostat command output from federated server

The **lspv** command was used to show paging space utilization on the federated server, as shown in Figure 4-11.

```
jamesbay.almaden.ibm.com - PuTTY
$ lspv -a
Page Space      Physical Volume  Volume Group    Size %Used Active  Auto  Type
hd6             hdisk0          rootvg         64512MB    1    yes   yes   lv
$
```

Figure 4-11 lspv command output from the federated server

The **ps -ef** and **ps aux** commands were also run to display the processes running on the federated server and their associated resource utilization, as shown in Figure 4-12 on page 175 and Figure 4-13 on page 175.

```

jamesbay.almaden.ibm.com - PuTTY
$ ps -ef | pg
  UID      PID      PPID  C   STIME   TTY   TIME CMD
  root         1         0  0   Jun 16   -   0:03 /etc/init
  root    73862    217186  0   Jun 16   -   0:00 /usr/sbin/portmap
  root    77902    163948  0   Jun 16   -   0:20 /usr/lpp/X11/bin/X -D /usr/lib
X11//rgb -T -force :0 -auth /var/dt/A:0-OX7gMa
  root    82008    217186  0   Jun 16   -   0:00 /usr/sbin/syslogd
  root    94300         1  0   Jun 16   -   0:00 /usr/lib/errdemon
  root   102502    163948  0   Jun 16   -   0:00 dtlogin <:0> -daemon
  root   106584    102502  0   Jun 16   -   0:15 /usr/dt/bin/dtsession
  root   114788         1  0   Jun 16   - 31:15 /usr/sbin/syncd 60
  root   123082    217186  0   Jun 16   -   0:00 /usr/sbin/nfsd 3891
  root   127102    208930  0   Jun 16   -   0:00 rpc.ttdbserver 100083 1
  root   135364    106584  0   Jun 16   -   0:22 dtwm
  root   139364    217186  0   Jun 16   -   0:07 /usr/sbin/aixmibd
  root   147560    217186  0   Jun 16   -   0:00 /usr/sbin/smmpd
  root   155736         1  0   Jun 16   -   0:00 /usr/ccs/bin/shlap64
  root   163948         1  0   Jun 16   -   0:00 /usr/dt/bin/dtlogin -daemon
  kawa   168180         1  0 13:52:43 -   0:00 /home/kawa/sql/lib/bin/db2bpf 25
4182A210 5 A

```

Figure 4-12 `ps -ef` command output from the federated server

```

jamesbay.almaden.ibm.com - PuTTY
$ ps aux | pg
USER      PID %CPU %MEM    SZ   RSS     TTY STAT   STIME   TIME COMMAND
root      32784 12.1  0.0   40    40     - A      Jun 16 10928:13 wait
root      36882 12.1  0.0   40    40     - A      Jun 16 10928:03 wait
root      24588 12.1  0.0   40    40     - A      Jun 16 10919:30 wait
root      20490 12.1  0.0   40    40     - A      Jun 16 10909:21 wait
root      28686 12.1  0.0   40    40     - A      Jun 16 10904:50 wait
root       8196 12.1  0.0   40    40     - A      Jun 16 10862:47 wait
root     12294 12.0  0.0   40    40     - A      Jun 16 10838:07 wait
root     16392 12.0  0.0   40    40     - A      Jun 16 10794:10 wait
kawa     2691082  0.6  0.0 14368 12392  - A      01:04:37 33:59 db2agent (FEDS
E
kawa     1576996  0.4  0.0 15488 13496  - A      01:04:38 24:13 db2agent (FEDS
E
kawa     1310878  0.4  0.0 13924 11916  - A      01:04:38 23:50 db2agent (FEDS
E
kawa       430086  0.4  0.0 15304 13244  - A      01:03:39 23:40 db2agent (FEDS
E
kawa     1351898  0.4  0.0 44104 42096  - A      01:04:38 22:54 db2agent (FEDS
E

```

Figure 4-13 `ps aux` command from federated server

The commands show:

- ▶ The **vmstat** command output in Figure 4-9 on page 173 has the CPU “id” column (which represents CPU idle percentage) ranging from 95 to 99 percent, indicating no CPU problems.
- ▶ The **iostat** command output in Figure 4-10 on page 174 has the “% tm_act” column (which lists the percentage of time the disk was busy representing bandwidth utilization) to be well below 40 percent, indicating no disk I/O utilization problems.
- ▶ The **lsps** command output in Figure 4-11 on page 174 has the “%Used” column (which lists the percent used of the paging space) to be 1 percent, indicating no paging problems.
- ▶ The **ps -ef** and **ps aux** command outputs (Figure 4-12 on page 175 and Figure 4-13 on page 175) have the CPU used for each process (“TIME” and “%CPU” columns, respectively) well within bounds, indicating no process CPU utilization problems.

Based on the above observations, we concluded that the performance problem was not related to the machine or operating system of the federated server machine.

Hypothesis 3: Federated database server performance

After confirming that the overall performance on the federated server machine looked acceptable, we looked at the DB2 II federated database server.

We looked at three potential system constraints areas that could affect query performance on the federated database server. These potential constraints areas are:

- ▶ Connection constraints
- ▶ Sorting constraints
- ▶ Buffer pool constraints

Each of these aspects are validated in the following subsections.

Connection constraints

To determine whether connection constraints are a factor, we ran DB2 commands to review the relevant settings of the database manager and database configuration parameters, as shown in Example 4-10 and Example 4-11 on page 177.

Example 4-10 DBM CFG parameter settings affecting connections

```
$ db2 get dbm cfg | grep -i agent
Priority of agents                (AGENTPRI) = SYSTEM
Max number of existing agents    (MAXAGENTS) = 200
```

Agent pool size	(NUM_POOLAGENTS) = 10
Initial number of agents in pool	(NUM_INITAGENTS) = 5
Max number of coordinating agents	(MAX_COORDAGENTS) = (MAXAGENTS - NUM_INITAGENTS)
Max no. of concurrent coordinating agents	(MAXCAGENTS) = MAX_COORDAGENTS
Max number of client connections	(MAX_CONNECTIONS) = MAX_COORDAGENTS
Number of pooled fenced processes	(FENCED_POOL) = MAX_COORDAGENTS
SPM resync agent limit	(SPM_MAX_RESYNC) = 20

Example 4-11 DB CFG parameter settings affecting connections

```
$db2 get db cfg for fedserv | grep -i appl
```

Catalog cache size (4KB)	(CATALOGCACHE_SZ) = (MAXAPPLS*4)
Max size of appl. group mem set (4KB)	(APPGROUP_MEM_SZ) = 30000
Percent of mem for appl. group heap	(GROUPHEAP_RATIO) = 70
Max appl. control heap size (4KB)	(APP_CTL_HEAP_SZ) = 128
Default application heap (4KB)	(APPLHEAPSZ) = 512
Package cache size (4KB)	(PCKCACHESZ) = (MAXAPPLS*8)
Percent. of lock lists per application	(MAXLOCKS) = 10
Max number of active applications	(MAXAPPLS) = 120
Average number of active applications	(AVG_APPLS) = 1
Max DB files open per application	(MAXFILOP) = 1024

We also collected snapshots showing monitor elements relevant to connection information. These snapshots are shown in Example 4-12 and Example 4-13 on page 178.

Example 4-12 DBM snapshot for connection information

```
$db2 get snapshot for dbm
```

Database Manager Snapshot

.....lines have been removed.....

Remote connections to db manager	= 12
Remote connections executing in db manager	= 0
Local connections	= 0
Local connections executing in db manager	= 0
Active local databases	= 1

High water mark for agents registered	= 199
High water mark for agents waiting for a token	= 0
Agents registered	= 68
Agents waiting for a token	= 0
Idle agents	= 0

.....lines have been removed.....

Agents assigned from pool	= 9584
Agents created from empty pool	= 14027
Agents stolen from another application	= 0
High water mark for coordinating agents	= 199
Max agents overflow	= 3
Hash joins after heap threshold exceeded	= 0

.....lines have been removed.....

Example 4-13 DB snapshot for connection information

\$db2 get snapshot for database on fedserv

Database Snapshot

.....lines have been removed.....

High water mark for connections	= 15
Application connects	= 8276
Secondary connects total	= 0
Applications connected currently	= 12
Appls. executing in db manager currently	= 0
Agents associated with applications	= 12
Maximum agents associated with applications	= 15
Maximum coordinating agents	= 15

.....lines have been removed.....

We concluded that connections were not the source of the performance problem for the following reasons.

The database manager configuration parameter settings in Example 4-10 on page 176 show the following:

- ▶ **Max number of existing agents (MAXAGENTS)** is 200.
- ▶ **Max number of client connections (MAX_CONNECTIONS)** is 195.
- ▶ **Max number of concurrent coordinating agents (MAXCAGENTS)** is 195.

The database configuration parameter settings in Example 4-11 on page 177 show the following: **Max number of active applications (MAXAPPLS)** for the fedserv database is 120.

The database manager snapshot monitor values in Example 4-12 on page 177 show the following:

- ▶ The sum of (**Remote connections to db manager** + **Local connections**) is 12, which is less than MAX_CONNECTIONS (195).

- ▶ The **High water mark for agents registered** (199) is less than MAXAGENTS (200).
- ▶ The **High water mark for agents waiting for a token** is 0, which is less than MAXCAGENTS (195).
- ▶ The **High water mark for coordinating agents** is 199, less than MAX_COORDAGENTS (195).

The database snapshot monitor values in Example 4-13 on page 178 show the following: **The High water mark for connections** is 15, less than MAXAPPLS (120).

Sorting constraints

To determine whether sorting constraints are a factor, we ran DB2 commands to review the relevant settings of the database manager and database configuration parameters, as shown in Example 4-14, Example 4-15, and Example 4-16.

Example 4-14 DB and DBM CFG parameters affecting sorting

```
$ db2 get db cfg for fedserv | grep -i sort
Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR) = (SHEAPTHRES)
Sort list heap (4KB) (SORTHEAP) = 20000
Index sort flag (INDEXSORT) = YES

$ db2 get dbm cfg | grep -i sort
Sort (DFT_MON_SORT) = ON
Sort heap threshold (4KB) (SHEAPTHRES) = 100000
```

Example 4-15 DBM snapshot for sorting information

```
$ db2 get snapshot for dbm | grep -i sort
Private Sort heap allocated = 0
Private Sort heap high water mark = 20018
Post threshold sorts = 0
Piped sorts requested = 4863
Piped sorts accepted = 4863
Sorting Information (SORT) = ON 06/21/2004 11:15:57.323761
```

Example 4-16 DB snapshot for sorting information

```
$ db2 get snapshot for db on fedserv | grep -i sort
Total Private Sort heap allocated = 0
Total Shared Sort heap allocated = 0
Shared Sort heap high water mark = 0
Total sorts = 4978
Total sort time (ms) = 58
Sort overflows = 1
Active sorts = 0
```

We concluded that sorts were not the cause of the performance problem for the following reasons.

The database manager snapshot monitor values in Example 4-15 on page 179 show the following:

- ▶ **Private Sort heap high water mark** (20018) is less than SHEAPTHRES (100000).
- ▶ **Post threshold sorts** is 0.
- ▶ The difference between **Piped sorts requested** and **Piped sorts accepted** is 0.

The database snapshot monitor values in Example 4-16 on page 179 show the following:

- ▶ **Shared Sort heap high water mark** is 0.
- ▶ **Sort overflows** is 1 and the average sort time is (58 / 4978), which is very small.

Buffer pool constraints

To determine whether buffer pool constraints are a factor, we ran DB2 commands to review the relevant settings of the database manager and database configuration parameters, as shown in Example 4-17.

Example 4-17 Buffer pool snapshot information

get snapshot for bufferpools on fedserv

Bufferpool Snapshot

Bufferpool name	= IBMDEFAULTBP
Database name	= FEDSERV
Database path	=
/data1/npart/db2i64/NODE0000/SQL00001/	
Input database alias	= FEDSERV
Snapshot timestamp	= 06/24/2004 17:27:11.418138
Buffer pool data logical reads	= 7908
Buffer pool data physical reads	= 135
Buffer pool temporary data logical reads	= 7
Buffer pool temporary data physical reads	= 0
Buffer pool data writes	= 0
Buffer pool index logical reads	= 32557
Buffer pool index physical reads	= 92
Buffer pool temporary index logical reads	= 0
Buffer pool temporary index physical reads	= 0
Total buffer pool read time (ms)	= 293

Total buffer pool write time (ms)	= 0
Asynchronous pool data page reads	= 9
Asynchronous pool data page writes	= 0
Buffer pool index writes	= 0
Asynchronous pool index page reads	= 0
Asynchronous pool index page writes	= 0
Total elapsed asynchronous read time	= 28
Total elapsed asynchronous write time	= 0
Asynchronous data read requests	= 4
Asynchronous index read requests	= 0
No victim buffers available	= 0
Direct reads	= 408
Direct writes	= 0
Direct read requests	= 57
Direct write requests	= 0
Direct reads elapsed time (ms)	= 0
Direct write elapsed time (ms)	= 0
Database files closed	= 0
Data pages copied to extended storage	= 0
Index pages copied to extended storage	= 0
Data pages copied from extended storage	= 0
Index pages copied from extended storage	= 0
Unread prefetch pages	= 0
Vectored I/Os	= 4
Pages from vectored I/Os	= 9
Block I/Os	= 0
Pages from block I/Os	= 0
Physical page maps	= 0
Node number	= 0
Tablespaces using bufferpool	= 5
Alter bufferpool information:	
Pages left to remove	= 0
Current size	= 75000
Post-alter size	= 75000

We concluded that the buffer pool was not the cause of the performance problem for the following reasons. The buffer pool snapshot monitor values in Example 4-17 on page 180 show the following:

- ▶ The buffer pool temporary data logical reads was 7, and temporary data physical reads was 0. Both values were very low and did not indicate any overflows for temporary data.
- ▶ The overall buffer pool hit ratio is $(1 - ((135 + 92) / (7908 + 32557))) * 100\%$, which is approximately 99 percent.
- ▶ The Index buffer pool hit ratio is $(1 - (92 / 32557)) * 100\%$, which is about 100 percent.

Attention: We have not shown our review of cache size constraints and miscellaneous constraints here, but determined that there were no system-wide performance constraints at the database server level using the procedures described in “Cache size constraints” on page 134 and “Miscellaneous constraints” on page 135, respectively.

Having eliminated system-wide federated server and database server resource constraints as the potential cause of the performance problem, the most likely cause was with a specific query.

Hypothesis 4: Federated application/query performance

Before one can investigate the cause of a query’s performance problem, one needs to identify the query in question. After identifying the query in question, one can begin diagnosing whether the performance problem is at the federated server, the remote data source, or equally divided between the two, as discussed in Example 4.2 on page 119.

1. Identify the application and query.

In some cases, we might know the application that is experiencing the performance problem because the users have complained about it specifically. However, the application may contain many queries, and it is necessary to pinpoint the specific query causing the performance problem before proceeding.

The approach used to pinpoint the problem query depends upon whether the query is currently running as discussed in E on page 121.

Example 4-18 is an application snapshot that contains information we can use to identify the specific queries executing at the time the snapshot is taken.

Example 4-18 Application snapshot

\$db2 get snapshot for all applications

Application Snapshot

Application handle	= 75
Application status	= Federated request pending
Status change time	= 06/24/2004 18:11:17.161566
Application code page	= 819
Application country/region code	= 1
DUOW correlation token	= *LOCAL.db2i64.0E07B5011047
Application name	= db2bp
Application ID	= *LOCAL.db2i64.0E07B5011047
Sequence number	= 0001
TP Monitor client user ID	=
TP Monitor client workstation name	=

```

TP Monitor client application name      =
TP Monitor client accounting string    =

Connection request start timestamp     = 06/24/2004 18:10:47.445195
Connect request completion timestamp   = 06/24/2004 18:10:47.467482
Application idle time                  =
CONNECT Authorization ID               = DB2I64
Client login ID                       = db2i64
Configuration NNAME of client          =
Client database manager product ID     = SQL08020
Process ID of client application       = 2220154
Platform of client application         = AIX 64BIT
Communication protocol of client       = Local Client

Inbound communication address          = *LOCAL.db2i64

```

.....lines have been removed.....

```

Statement type                        = Dynamic SQL Statement
Statement                            = Fetch
Section number                       = 201
Application creator                   = NULLID
Package name                         = SQLC2E05
Consistency Token                    = AAAAaCDU
Package Version ID                   =
Cursor name                          = SQLCUR201
Statement database partition number   = 0
Statement start timestamp             = 06/24/2004 18:11:17.092428
Statement stop timestamp              =
Elapsed time of last completed stmt(sec.ms)= 0.000029
Total Statement user CPU time         = 0.010000
Total Statement system CPU time       = 0.010000

```

.....lines have been removed.....

Dynamic SQL statement text:

```
select c_name, sum(c_acctbal) from ora.customer group by c_name
```

.....lines have been removed.....

Example 4-18 on page 182 shows Dynamic SQL statement text that we have highlighted. Although there may be many statements executing at a time, application snapshot information may help narrow down the poorly performing statement if you are familiar with the application name, specific connection time, or the tables referenced in the statement text. The Elapsed time of last completed stmt (sec.ms), Total Statement user CPU time, and Total Statement system CPU time fields in the snapshot output can help identify poorly performing queries.

Since the performance problem was ongoing at the time, we reviewed the Dynamic SQL statement text in the application snapshot and used our knowledge of the tables accessed by the application to identify the poorly performing query shown in Example 4-19. This query reports the sum of account balances for all customers by customer name, and includes a SUM function and a GROUP BY clause.

Example 4-19 Problem query

```
select c_name, sum(c_acctbal)
from ora.customer
group by c_name;
```

2. Is the problem at the federated server or the remote data source or both?

Once the problem query has been identified, we need to determine if the query's performance problem is at the federated server, the remote data source, or distributed between both.

This information can be determined from a dynamic SQL snapshot, as discussed in 4.2.3, "Federated server or remote data source" on page 136. Example 4-20 is a dynamic SQL snapshot, and includes the problem query (highlighted) in the Statement text field.

Example 4-20 Dynamic SQL snapshot

```
$db2 get snapshot for dynamic sql on fedserv

      Dynamic SQL Snapshot Result

Database name              = FEDSERV

Database path              = /data1/npart/db2i64/NODE0000/SQL00001/

..... lines have been removed .....

Number of executions       = 1
Number of compilations     = 1
Worst preparation time (ms) = 80
Best preparation time (ms) = 80
Internal rows deleted      = 0
Internal rows inserted     = 0
Rows read                 = 1500002
Internal rows updated      = 0
Rows written               = 1500000
Statement sorts         = 1
Statement sort overflows = 1
```

```

Total sort time = 4019
Buffer pool data logical reads = 4
Buffer pool data physical reads = 1
Buffer pool temporary data logical reads = 31581
Buffer pool temporary data physical reads = 0
Buffer pool index logical reads = 6
Buffer pool index physical reads = 3
Buffer pool temporary index logical reads = 0
Buffer pool temporary index physical reads = 0
Total execution time (sec.ms) = 26.668711
Total user cpu time (sec.ms) = 11.510000
Total system cpu time (sec.ms) = 0.830000
Statement text = select c_name, sum(c_acctbal) from
ora.customer group by c_name

..... lines have been removed .....

Number of executions = 1
Number of compilations = 1
Worst preparation time (ms) = 0
Best preparation time (ms) = 0
Internal rows deleted = 0
Internal rows inserted = 0
Rows read = 1500000
Internal rows updated = 0
Rows written = 0
Statement sorts = 0
Buffer pool data logical reads = 0
Buffer pool data physical reads = 0
Buffer pool temporary data logical reads = 0
Buffer pool temporary data physical reads = 0
Buffer pool index logical reads = 0
Buffer pool index physical reads = 0
Buffer pool temporary index logical reads = 0
Buffer pool temporary index physical reads = 0
Total execution time (sec.ms) = 14.241853
Total user cpu time (sec.ms) = 0.000000
Total system cpu time (sec.ms) = 0.000000
Statement text = SELECT A0."C_NAME", A0."C_ACCTBAL" FROM
"IITEST"."CUSTOMER" A0

```

As discussed in 4.2.3, “Federated server or remote data source” on page 136, since there is no direct mechanism to link the remote SQL fragments in the dynamic cache with their corresponding user SQL statement, we generated **db2exfmt** output for the user query, as shown in Example 4-21, to obtain this link.

Example 4-21 db2exfmt output for the problem query

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

***** EXPLAIN INSTANCE *****

DB2_VERSION: 08.02.0
SOURCE_NAME: SQLC2E05
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-08-03-14.50.02.089128
EXPLAIN_REQUESTER: DB2I64

Database Context:

Parallelism: None
CPU Speed: 5.392596e-07
Comm Speed: 100
Buffer Pool size: 75000
Sort Heap size: 20000
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 640

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 203 -----

QUERYNO: 1
QUERYTAG:
Statement Type: Select
Updatable: No
Deletable: No
Query Degree: 1

Original Statement:

```

select c_name, sum(c_acctbal)
from ora.customer
group by c_name

```

Optimized Statement:

```

-----
SELECT Q3.$C0 AS "C_NAME", Q3.$C1
FROM
  (SELECT Q2.$C0, SUM(Q2.$C1)
   FROM
     (SELECT Q1.C_NAME, Q1.C_ACCTBAL
      FROM ORA.CUSTOMER AS Q1) AS Q2
   GROUP BY Q2.$C0) AS Q3

```

Access Plan:

```

-----
Total Cost: 126862
Query Degree:1

```

```

      Rows
RETURN
(   1)
Cost
  I/O
  |
300000
GRPBY
(   2)
126790
32173
  |
300000
TBSCAN
(   3)
126749
32173
  |
300000
SORT
(   4)
126574
32173
  |
300000
SHIP
(   5)
126102
32173

```

|
300000
NICKNM: ORA
CUSTOMER

1) RETURN: (Return Result)
Cumulative Total Cost: 126862
Cumulative CPU Cost: 1.80948e+09
Cumulative I/O Cost: 32173
Cumulative Re-Total Cost: 396.853
Cumulative Re-CPU Cost: 7.35922e+08
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 126574
Estimated Bufferpool Buffers: 0
Remote communication cost:165498

Arguments:

BLDLEVEL: (Build level)
DB2 v8.1.1.64 : s040509
ENVVAR : (Environment Variable)
DB2_EXTENDED_OPTIMIZATION = ON
STMTHEAP: (Statement heap size)
8192

Input Streams:

5) From Operator #2

Estimated number of rows: 300000
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q4.\$C1+Q4.C_NAME

2) GRPBY : (Group By)
Cumulative Total Cost: 126790
Cumulative CPU Cost: 1.67597e+09
Cumulative I/O Cost: 32173
Cumulative Re-Total Cost: 324.862
Cumulative Re-CPU Cost: 6.02422e+08
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 126574

Estimated Bufferpool Buffers: 0
Remote communication cost:165498

Arguments:

AGGMODE : (Aggregation Mode)
COMPLETE
GROUPBYC: (Group By columns)
TRUE
GROUPBYN: (Number of Group By columns)
1
GROUPBYR: (Group By requirement)
1: Q2.C_NAME
ONEFETCH: (One Fetch flag)
FALSE

Input Streams:

4) From Operator #3

Estimated number of rows: 300000
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.C_NAME(A)+Q2.C_ACCTBAL

Output Streams:

5) To Operator #1

Estimated number of rows: 300000
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q4.\$C1+Q4.C_NAME

3) TBSCAN: (Table Scan)

Cumulative Total Cost: 126749
Cumulative CPU Cost: 1.60097e+09
Cumulative I/O Cost: 32173
Cumulative Re-Total Cost: 284.417
Cumulative Re-CPU Cost: 5.27421e+08
Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 126574
Estimated Bufferpool Buffers: 0
Remote communication cost:165498

Arguments:

MAXPAGES: (Maximum pages for prefetch)
ALL
PREFETCH: (Type of Prefetch)
NONE
SCANDIR : (Scan Direction)
FORWARD

Input Streams:

3) From Operator #4

Estimated number of rows: 300000
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.C_NAME(A)+Q2.C_ACCTBAL

Output Streams:

4) To Operator #2

Estimated number of rows: 300000
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.C_NAME(A)+Q2.C_ACCTBAL

4) SORT : (Sort)

Cumulative Total Cost: 126574
Cumulative CPU Cost: 1.27517e+09
Cumulative I/O Cost: 32173
Cumulative Re-Total Cost: 108.725
Cumulative Re-CPU Cost: 2.0162e+08
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 126574
Estimated Bufferpool Buffers: 32173
Remote communication cost:165498

Arguments:

DUPLWARN: (Duplicates Warning flag)
FALSE
NUMROWS : (Estimated number of rows)
300000
ROWWIDTH: (Estimated width of rows)
40
SORTKEY : (Sort Key column)
1: Q2.C_NAME(A)
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

2) From Operator #5

Estimated number of rows: 300000
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.C_ACCTBAL+Q2.C_NAME

Output Streams:

3) To Operator #3

Estimated number of rows: 300000
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.C_NAME(A)+Q2.C_ACCTBAL

5) SHIP : (Ship)

Cumulative Total Cost: 126102
Cumulative CPU Cost: 4.00297e+08
Cumulative I/O Cost: 32173
Cumulative Re-Total Cost: 108.725
Cumulative Re-CPU Cost: 2.0162e+08
Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 25.0079
Estimated Bufferpool Buffers: 32173
Remote communication cost:165498

Arguments:

CSERQY : (Remote common subexpression)
FALSE
DSTSEVER: (Destination (ship to) server)
- (NULL).
RMTQTX : (Remote statement)
SELECT AO."C_NAME", AO."C_ACCTBAL" FROM "IITEST"."CUSTOMER" AO
SRCSEVER: (Source (ship from) server)
ORASERV
STREAM : (Remote stream)
FALSE

Input Streams:

1) From Object ORA.CUSTOMER

Estimated number of rows: 300000
Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:

+Q1.\$RID\$+Q1.C_ACCTBAL+Q1.C_NAME

Output Streams:

2) To Operator #4

Estimated number of rows: 300000
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.C_ACCTBAL+Q2.C_NAME

Objects Used in Access Plan:

Schema: DB2I64
Name: ORDERSMQT
Type: Materialized View (reference only)

Schema: ORA
Name: CUSTOMER
Type: Nickname
Time of creation: 2004-07-08-17.26.10.941686
Last statistics update: 2004-08-03-14.48.39.498055
Number of columns: 8
Number of rows: 300000
Width of rows: 63
Number of buffer pool pages: 32173
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

The RMTQTX field of the SHIP operator 5 in Example 4-21 on page 186 contains the remote SQL fragment:

```
SELECT AO."C_NAME", AO."C_ACCTBAL" FROM "IITEST"."CUSTOMER" AO
```

This remote SQL fragment is located in the dynamic cache output shown in Example 4-20 on page 184, and the following information can be gathered:

- **Number of executions** is 1 for the user-entered query as well as the single remote SQL fragment.
- **Total execution time (sec.ms)**, which is 26.668711 for the user-entered query, and 14.241853 for the remote SQL fragment.
- **Total user cpu time (sec.ms)** and **Total system cpu time (sec.ms)**, which is 11.510000 and 0.830000 for the user-entered query, and zero for the remote SQL fragment.
- **Rows read** is 1500002 for the user-entered query, which is the number of rows returned to the user; and 1500000 for the remote SQL fragment, which indicates the number of rows returned to the federated server from the remote data source.
- Other fields of interest include **Statement sorts**, **Statement sort overflows**, and **Total sort time**, which only apply to the user-entered query. They have values 1, 1, and 4019, respectively.

Note: To obtain the average elapsed and CPU times, as well as the number of rows returned, you must divide the values shown by the **Number of executions**.

We can derive the following information from these metrics:

- The average number of rows returned from the remote data source to the federated server is $(1500000 / 1) = 1500000$.
- The average elapsed time for the user query is $(26.668711 / 1) = 26.668711$ seconds, while that of the remote SQL fragment is $(14.241853 / 1) = 14.241853$ seconds.

Attention: In our example, given that the total query elapsed time is 26.668711 seconds, it is clear that the elapsed time of the query is spent more or less equally at the federated server and the remote data source. The percentage of time spent at the remote data source is $((14.241853 / 26.668711) \times 100) = 53.4\%$.

If the information for the user query had not been available from the dynamic cache, SQL snapshot, you could possibly re-run the query in **db2batch** to get similar statistics, as shown in Example 4-22 and Example 4-23.

Example 4-22 db2batch command

```
$db2batch -d fedserv -f query1.sql -r query1.out -o r 10 p 3
```

Example 4-23 db2batch output

```
select c_name, sum(c_acctbal)
from ora.customer
group by c_name
```

C_NAME	2

Customer#000000001	711.56
Customer#000000002	121.65
Customer#000000003	7498.12
Customer#000000004	2866.83
Customer#000000005	794.47
Customer#000000006	7638.57
Customer#000000007	9561.95
Customer#000000008	6819.74
Customer#000000009	8324.07
Customer#000000010	2753.54

Number of rows retrieved is: 1500000
Number of rows sent to output is: 10

Elapsed Time is: 25.760 seconds

Locks held currently = 0
Lock escalations = 0


```

Total sorts                                     = 1
Total sort time (ms)                           = 4029
Sort overflows                                 = 1
Buffer pool data logical reads                 = 0
Buffer pool data physical reads                = 0
Buffer pool data writes                       = 0
Buffer pool index logical reads                = 0
Buffer pool index physical reads               = 0
Buffer pool index writes                      = 0
Total buffer pool read time (ms)               = 0
Total buffer pool write time (ms)              = 0
Asynchronous pool data page reads             = 0
Asynchronous pool data page writes            = 0
Asynchronous pool index page reads            = 0
Asynchronous pool index page writes           = 0
Total elapsed asynchronous read time           = 0
Total elapsed asynchronous write time          = 0
Asynchronous read requests                    = 0
LSN Gap cleaner triggers                      = 0
Dirty page steal cleaner triggers              = 0
Dirty page threshold cleaner triggers          = 0
Direct reads                                  = 0
Direct writes                                  = 0
Direct read requests                          = 0
Direct write requests                         = 0
Direct read elapsed time (ms)                 = 0
Direct write elapsed time (ms)                = 0
Rows selected                                 = 1500000
Log pages read                                = 0
Log pages written                              = 0
Catalog cache lookups                         = 0
Catalog cache inserts                        = 0
Buffer pool data pages copied to ext storage  = 0
Buffer pool index pages copied to ext storage = 0
Buffer pool data pages copied from ext storage = 0
Buffer pool index pages copied from ext storage = 0
Total Agent CPU Time (seconds)                 = 12.05
Post threshold sorts                          = 0
Piped sorts requested                         = 88
Piped sorts accepted                          = 88
-----

```

Summary of Results

=====

Statement #	Elapsed Time (s)	Agent CPU Time (s)	Rows Fetched	Rows Printed
1	25.760	12.050	1500000	10

Arith. mean	25.760	12.1
Geom. mean	25.760	12

Since the processing time is consumed almost equally at the federated server and the remote data source, we need to investigate both federated server and the remote data source related items, as described in 4.2.5, “Remote data source related” on page 161.

3. Federated server related.

The **Total sort time** field in the dynamic cache output for the user-entered query in Example 4-20 on page 184 is 4019 milliseconds, while the (**Total user cpu time (sec.ms)** + **Total system CPU time (sec.ms)**) is = $(11.510000 + 0.830000) = 12.340000$ seconds. This is almost equal to the $(26.668711 - 14.241853) = 12.425257$ seconds spent at the federated server and is probably entirely due to the processing of 1500002 rows. Unless the number of rows is reduced, the processing time at the federated server can not be reduced.

4. Remote data source related.

The **Rows read** field in the dynamic cache output for the remote SQL fragment in Example 4-20 on page 184 has 1500000 rows, while the SHIP operator 5 in the Access Plan section of Example 4-21 on page 186 estimates that 300000 rows would be returned to the federated server. The Objects used in the Access Plan section of Example 4-21 on page 186 also shows the Number of rows being 300000 for the ORA.CUSTOMER nickname. This information is obtained from nickname statistics recorded in the global catalog.

Root cause of the problem

It appears that the actual number of rows retrieved from the remote data source (1500000) has increased significantly as compared to the statistics stored in the global catalog (300000), and is the primary reason for the extended response times.

Apply best practices

We recommend the following best practices to address the problem of a large number of rows being returned to the federated server from the remote data source:

- Ensure that the federated server’s global catalog is completely in sync with the remote data source catalog information—both statistics and index information.

Important: We strongly recommend that you do *not* drop and recreate the nickname to achieve synchronization, since it can have undesirable effects on dependant objects such as dropping MQTs and marking views as inoperable.

This involves the following steps:

- a. Have the DBA at the remote data source update the catalog by running the appropriate command/utility for ORA.CUSTOMER. The Oracle command to update statistics is:

analyze table <tablename> compute statistics
- b. Synchronize the federated server global catalog with statistics and index information. This could involve a combination of executing the **NNSTAT** stored procedure (see Example 4-24), and manually synchronizing index specifications and nickname statistics as discussed in “Statistics” on page 152.

Example 4-25 shows a query for determining the indexes associated with the nickname ORA.CUSTOMER.

Example 4-24 Update nickname statistics from command line

```
connect to fedserv;  
CALL SYSPROC.NNSTAT('FEDSERV', 'ORA', 'CUSTOMER', '/home/db2i64', ?,  
?);
```

Example 4-25 Nickname index specifications

```
connect to fedserv
```

Database Connection Information

```
Database server      = DB2/AIX64 8.2.0  
SQL authorization ID = DB2I64  
Local database alias = FEDSERV
```

```
select substr(tabschema,1,4) as tabschema, substr(tabname,1,8) as tabname,  
substr(indschema,1,4) as indschema, uniquerule, substr(indname,1,16) as  
indname, substr(colnames,1,30) as colnames  
from syscat.indexes  
where tabschema = 'ORA' and tabname = 'CUSTOMER'
```

TABSCHEMA	TABNAME	INDSCHEMA	INDNAME	UNIQUERULE	COLNAMES
ORA	CUSTOMER	ORA	C_CHK	P	+C_CUSTKEY
ORA	CUSTOMER	ORA	C_MS_CHK	U	+C_MKTSEGMENT+C_CUSTKEY

```
ORA          CUSTOMER ORA          C_NAT_CKEY_RE  U          +C_NATIONKEY+C_CUSTKEY
```

```
3 record(s) selected.
```

Investigations revealed that the indexes were in sync, and no new index specifications needed to be created or existing indexes dropped.

- Execute **db2exfmt** with the current statistics and review access path changes, if any. Example 4-26 shows that there were no changes to the access path. If there are changes, consider executing the query to assess performance.

Example 4-26 db2exfmt after nickname statistics updated

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

```
***** EXPLAIN INSTANCE *****
```

```
DB2_VERSION: 08.02.0
SOURCE_NAME: SQLC2E05
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-07-08-17.26.19.195544
EXPLAIN_REQUESTER: DB2I64
```

Database Context:

```
-----
Parallelism: None
CPU Speed: 5.392596e-07
Comm Speed: 100
Buffer Pool size: 75000
Sort Heap size: 20000
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 640
```

Package Context:

```
-----
SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability
```

----- STATEMENT 1 SECTION 203 -----

QUERYNO: 1
QUERYTAG:
Statement Type: Select
Updatable: No
Deletable: No
Query Degree: 1

Original Statement:

select c_name, sum(c_acctbal)
from ora.customer
group by c_name

Optimized Statement:

SELECT Q3.\$C0 AS "C_NAME", Q3.\$C1
FROM
 (SELECT Q2.\$C0, SUM(Q2.\$C1)
 FROM
 (SELECT Q1.C_NAME, Q1.C_ACCTBAL
 FROM ORA.CUSTOMER AS Q1) AS Q2
 GROUP BY Q2.\$C0) AS Q3

Access Plan:

Total Cost: 131549
Query Degree:1

Rows
RETURN
(1)
Cost
I/O
|
1.5e+06
GRPBY
(2)
131189
32173
|
1.5e+06
TBSCAN
(3)
130987
32173
|

```

1.5e+06
SORT
( 4)
130105
32173
|
1.5e+06
SHIP
( 5)
126494
32173
|
1.5e+06
NICKNM: ORA
CUSTOMER

```

```

1) RETURN: (Return Result)
Cumulative Total Cost: 131549
Cumulative CPU Cost: 1.05014e+10
Cumulative I/O Cost: 32173
Cumulative Re-Total Cost: 1443.68
Cumulative Re-CPU Cost: 2.67715e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 130105
Estimated Bufferpool Buffers: 16130
Remote communication cost:827474

```

Arguments:

```

-----
BLDLEVEL: (Build level)
DB2 v8.1.1.64 : s040509
ENVVAR : (Environment Variable)
DB2_EXTENDED_OPTIMIZATION = ON
STMHEAP: (Statement heap size)
8192

```

Input Streams:

```

-----
5) From Operator #2

```

```

Estimated number of rows: 1.5e+06
Number of columns: 2
Subquery predicate ID: Not Applicable

```

```

Column Names:
-----

```

+Q4.\$C1+Q4.C_NAME

2) GRPBY : (Group By)

Cumulative Total Cost: 131189
Cumulative CPU Cost: 9.83393e+09
Cumulative I/O Cost: 32173
Cumulative Re-Total Cost: 1083.72
Cumulative Re-CPU Cost: 2.00965e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 130105
Estimated Bufferpool Buffers: 16130
Remote communication cost:827474

Arguments:

AGGMODE : (Aggregation Mode)
COMPLETE
GROUPBYC: (Group By columns)
TRUE
GROUPBYN: (Number of Group By columns)
1
GROUPBYR: (Group By requirement)
1: Q2.C_NAME
ONEFETCH: (One Fetch flag)
FALSE

Input Streams:

4) From Operator #3

Estimated number of rows: 1.5e+06
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.C_NAME(A)+Q2.C_ACCTBAL

Output Streams:

5) To Operator #1

Estimated number of rows: 1.5e+06
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q4.C1+Q4.C_NAME

3) TBSCAN: (Table Scan)
Cumulative Total Cost: 130987
Cumulative CPU Cost: 9.45893e+09
Cumulative I/O Cost: 32173
Cumulative Re-Total Cost: 881.499
Cumulative Re-CPU Cost: 1.63465e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 130105
Estimated Bufferpool Buffers: 16130
Remote communication cost:827474

Arguments:

MAXPAGES: (Maximum pages for prefetch)
ALL
PREFETCH: (Type of Prefetch)
NONE
SCANDIR : (Scan Direction)
FORWARD

Input Streams:

3) From Operator #4

Estimated number of rows: 1.5e+06
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.C_NAME(A)+Q2.C_ACCTBAL

Output Streams:

4) To Operator #2

Estimated number of rows: 1.5e+06
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.C_NAME(A)+Q2.C_ACCTBAL

4) SORT : (Sort)
Cumulative Total Cost: 130105
Cumulative CPU Cost: 7.82428e+09
Cumulative I/O Cost: 32173
Cumulative Re-Total Cost: 0
Cumulative Re-CPU Cost: 0
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 130105
Estimated Bufferpool Buffers: 48303
Remote communication cost:827474

Arguments:

DUPLWARN: (Duplicates Warning flag)
FALSE
NUMROWS : (Estimated number of rows)
1500000
ROWWIDTH: (Estimated width of rows)
40
SORTKEY : (Sort Key column)
1: Q2.C_NAME(A)
SPILLED : (Pages spilled to bufferpool or disk)
16130
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

2) From Operator #5

Estimated number of rows: 1.5e+06
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.C_ACCTBAL+Q2.C_NAME

Output Streams:

3) To Operator #3

Estimated number of rows: 1.5e+06
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.C_NAME(A)+Q2.C_ACCTBAL

5) SHIP : (Ship)
Cumulative Total Cost: 126494
Cumulative CPU Cost: 1.12815e+09
Cumulative I/O Cost: 32173
Cumulative Re-Total Cost: 501.226
Cumulative Re-CPU Cost: 9.29471e+08
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 25.0079
Estimated Bufferpool Buffers: 32173
Remote communication cost:827474

Arguments:

CSERQY : (Remote common subexpression)
FALSE
DSTSEVER: (Destination (ship to) server)
- (NULL).
RMTQTX : (Remote statement)
SELECT A0."C_NAME", A0."C_ACCTBAL" FROM "IITEST"."CUSTOMER" A0
SRCSEVER: (Source (ship from) server)
ORASERV
STREAM : (Remote stream)
FALSE

Input Streams:

1) From Object ORA.CUSTOMER

Estimated number of rows: 1.5e+06
Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:

+Q1.\$RID\$+Q1.C_ACCTBAL+Q1.C_NAME

Output Streams:

2) To Operator #4

Estimated number of rows: 1.5e+06
Number of columns: 2

Subquery predicate ID: Not Applicable

Column Names:

+Q2.C_ACCTBAL+Q2.C_NAME

Objects Used in Access Plan:

Schema: ORA

Name: CUSTOMER

Type: Nickname

Time of creation: 2004-07-08-17.26.10.941686

Last statistics update:

Number of columns: 8

Number of rows: 1500000

Width of rows: 63

Number of buffer pool pages: 32173

Distinct row values: No

Tablespace name:

Tablespace overhead: 24.100000

Tablespace transfer rate: 0.900000

Source for statistics: Single Node

Prefetch page count: 32

Container extent page count: 32

-
- If there are no changes (as was the situation with our problem query) then consider:
 - Resetting user expectations in the light of the increased volume of data
 - Asking the remote DBA to tune their environment for the SQL fragment
 - Investigating the possibility of encouraging pushdown of the GROUP BY to reduce the number of rows being returned to the federated server by setting server options affecting the collating sequence
 - Investigating the possibility of designing MQTs on the nickname in order to improve query performance

Note: Tuning the sort heap and buffer pool does not seem appropriate for this query since the sort costs are quite small relative to the cost of retrieving rows from the remote data source.

Since tuning tends to be a trial-and-error iterative process, it is necessary to continue to monitor the system in order to assess the impact of the changes and act accordingly.

4.4.3 Poorly tuned sort heap and buffer pools

The SORTHEAP database configuration parameter and the buffer pool for the temporary tablespace are system-wide tuning options that can impact the performance of a federated query. Potential performance problems with them are generally detected through routine monitoring of the DB2 II environment.

Federated servers can have significant requirements for sorting rows due to joins, ORDER BY, and GROUP BY activity. Sort performance can be impacted by SORTHEAP, SHEAPTHRES, the setting of INTRA_PARALLEL, and buffer pool size associated with temporary tablespaces.

In this scenario, we highlight the potential need for tuning the SORTHEAP and buffer pools in a DB2 II environment. Routine monitoring tends to be preemptive in nature since it involves detecting deteriorating trends of key performance drivers and addressing them before a problem gets out of hand.

Triggering event

Analysis of database manager and database snapshot activity highlighted potential problems of a high percentage of sort overflows meriting further investigation.

Hypotheses and validations

Since our triggering event was routine monitoring snapshots, we could go directly to the step of evaluating the federated database server, bypassing network performance and federated server performance considerations.

Example 4-27 and shows the snapshot output that triggered the investigation.

Example 4-27 Routine monitoring snapshot information

```
$db2 get snapshot for all on fedserv
```

Database Snapshot

Database name	= FEDSERV
Database path	=
/data1/npart/db2i64/NODE0000/SQL00001/	
Input database alias	= FEDSERV
Database status	= Active
Catalog database partition number	= 0
Catalog network node name	=
Operating system running at database server=	AIX 64BIT
Location of the database	= Local
First database connect timestamp	= Not Collected
Last reset timestamp	= Not Collected
Last backup timestamp	= Not Collected

Snapshot timestamp = 07/06/2004 18:57:24.781788

..... lines have been removed

Total Private Sort heap allocated = 1381
Total Shared Sort heap allocated = 0
Shared Sort heap high water mark = 0
Total sorts = **138036**
Total sort time (ms) = 268904
Sort overflows = **6914**
Active sorts = 5
..... lines have been removed

Bufferpool Snapshot

Bufferpool name = IBMDEFAULTBP
Database name = FEDSERV
Database path =
/data1/npart/db2i64/NODE0000/SQL00001/
Input database alias = FEDSERV
Snapshot timestamp = 07/06/2004 18:57:24.781788

Buffer pool data logical reads = 563
Buffer pool data physical reads = 166
Buffer pool temporary data logical reads = **3948935**
Buffer pool temporary data physical reads = **264347**
Buffer pool data writes = **329678**
Buffer pool index logical reads = 520
Buffer pool index physical reads = 124
Buffer pool temporary index logical reads = 0
Buffer pool temporary index physical reads = 0
Total buffer pool read time (ms) = 0
Total buffer pool write time (ms) = 0

Example 4-27 on page 206 shows the following sort and buffer pool related snapshot information of interest:

- The ratio ((**Sort overflows**) / (**Total sorts**)) is (6914 / 138036) = 5%.

This ratio had been rising steadily over an extended period of time and the current high value of 5 percent is considered to merit further investigation in order to preempt future user complaints about response times.

When sort overflows occur, data is written to the temporary tablespaces that are associated with a buffer pool. When the buffer pool fills, data is overflowed to disk. This can result in poor query performance.

- ▶ The **Buffer pool temporary data logical reads** (3948935) and **Buffer pool temporary data physical reads** (264347) values for the temporary table space IBMDEFAULTBP indicate a buffer hit ratio of $(1 - (264347 / 3948935)) = 93.1\%$, which is very good.
- ▶ The **Buffer pool data writes** (329678) value merits investigation since the large value indicates sort overflows to disk or temporary tables being created.

Hypothesis 1: Federated database server performance

Since we felt that the percentage of sort overflows was unacceptably high, we listed the following:

- ▶ Sort-related current database manager configuration and database configuration information, as shown in Example 4-28
- ▶ Size of the default buffer pool that was used for the temporary tablespace, as shown in Example 4-29
- ▶ Database manager snapshot output shown in Example 4-30 on page 208 that was captured at the same time as the database snapshot shown in Example 4-27 on page 206

Example 4-28 DB and DBM CFG parameters affecting sorting

```
$ db2 get db cfg for fedserv | grep -i sort
Sort heap thres for shared sorts (4KB) (SHEAPTHRES_SHR) = (SHEAPTHRES)
Sort list heap (4KB) (SORTHEAP) = 256
Index sort flag (INDEXSORT) = YES
$ db2 get dbm cfg | grep -i sort
Sort (DFT_MON_SORT) = ON
Sort heap threshold (4KB) (SHEAPTHRES) = 20000
```

Example 4-29 Default BP size

```
select bpname, npages, pagesize from syscat.bufferpools
```

BPNAME	NPAGES	PAGESIZE
IBMDEFAULTBP	1000	4096

1 record(s) selected.

Example 4-30 DBM snapshot for sorting information

```
$ db2 get snapshot for dbm | grep -i sort
```

```
Private Sort heap allocated = 1716
```

Private Sort heap high water mark	= 3055
Post threshold sorts	= 0
Piped sorts requested	= 138469
Piped sorts accepted	= 138469
Sorting Information	(SORT) = ON 07/06/2004 18:59:24.335670

Example 4-28 on page 208 shows the SORTHEAP and SHEAPTHRES parameters having default values of 256 and 20000, respectively, while the IBMDEFAULTBP buffer pool size (see Example 4-29 on page 208) was also set to the default value of 1000. It appears that these parameters have not been tuned for this particular workload.

The database manager snapshot (Example 4-30) shows the following information of interest:

- ▶ **Private Sort heap high water mark** is 3055.
- ▶ **Post threshold sorts** is 0.
- ▶ Ratio of **Piped sorts requested** and **Piped sorts accepted** is 100 percent.

There appear to be no problems with post threshold sorts or piped sorts. The Private Sort heap high water mark provides guidelines for setting the SHEAPTHRES database manager configuration parameter.

Root cause of the problem

Letting the SORTHEAP and SHEAPTHRES configuration parameters default appears to be at least one of the causes that could result in performance problems in the future. It is also possible that inadequate indexes as well as poorly written queries are causing sorts to place undue demand on the sort heap and the buffer pool.

Apply best practices

We recommend the following best practices to address the sort overflow problem on the DB2 II federated database server:

- ▶ Tune the SORTHEAP and/or SHEAPTHRES configuration parameters to minimize the percent of sort overflows. Increase these values incrementally via a trial-and-error process to minimize sort overflows. Example 4-31 shows the commands for modifying these parameters.

Example 4-31 Adjust SHEAPTHRES and SORTHEAP configuration parameters

```
update dbm cfg using sheapthres 100000
DB20000I The UPDATE DATABASE MANAGER CONFIGURATION command completed
successfully.
```

SQL1362W One or more of the parameters submitted for immediate modification were not changed dynamically. Client changes will not be effective until the next time the application is started or the TERMINATE command has been issued. Server changes will not be effective until the next DB2START command.

```
update db cfg for fedserv using sortheap 20000
DB20000I The UPDATE DATABASE CONFIGURATION command completed successfully.
```

- ▶ Monitor queries in the workload to determine if sorts can be eliminated or the number of rows sorted reduced, as follows:
 - If there are MQTs or local tables involved in a federated query, consider creating appropriate indexes on them to avoid sorts.
 - Verify that the DB2 II configuration options do not inhibit pushdown of predicates or sorts to the remote data source.
 - Facilitate the selection of hash join over merge scan join by ensuring that equality predicates have matching data types and matching scale and precision. This can minimize sorts even though the sort heap is used for the hash join.

Note: Tuning the buffer pool does not seem appropriate for this query since the hit ratio is sufficiently high (93.1 percent).

Since tuning tends to be a trial-and-error iterative process, it is necessary to continue to monitor the system in order to assess the impact of the changes and act accordingly.

4.4.4 Missing or unavailable MQTs

MQTs were designed to improve the performance of queries in a data warehousing environment where users often issue queries repetitively against large volumes of data with minor variations in a query's predicates. However, MQTs do not require an aggregate function in their definition to be beneficial.

MQTs provide a look-aside capability for such queries that can result in orders of magnitude improvement in performance. When appropriate MQTs are defined on base tables, queries that access these base tables are automatically rewritten (if appropriate) by the DB2 optimizer to access the MQTs instead, in order to achieve superior query performance.

An MQT can also be built against a federated nickname in addition to local tables on the federated server. Queries written against federated nicknames can be rewritten by the DB2 optimizer to choose the MQT instead to satisfy the query, thereby avoiding costly remote data source access.

In this scenario, we show how dropping and recreating a nickname in order to synchronize statistics and index information between the remote data source and the DB2 II global catalog can result in the dropping of an MQT dependent on this nickname. This in turn will result in performance degradation of queries that previously routed the query to the MQT. In DB2 II terminology, this is called caching the remote data locally in persistent tables (MQTs) and refreshing the data periodically to meet the currency needs of the users.

Triggering event

A user complained about poor response times with a specific query. The user claimed that the query had been performing okay until today.

Hypotheses and validations

Since this was a complaint from a user about the performance of a specific query, as per Figure 4-3 on page 122, we decided to enter the DB2 hypotheses hierarchy shown in Figure 4-1 on page 117 and described in Example 4.2 on page 119, at a lower level, bypassing network- and system-related problems, and focused directly on federated application/query performance, as follows:.

Hypothesis 1: Federated application/query performance

Before one can investigate the cause of a query's performance problem, one needs to identify the query in question. After identifying the query in question, one can begin diagnosing whether the performance problem is at the federated server, the remote data source, or equally divided between the two, as discussed in Example 4.2 on page 119.

1. Identify the application and query.

In this case the user specifically identified the query in question as being the one shown in Example 4-32. It counts by region the number of orders received in a particular period.

Example 4-32 Problem query

```
select r_name, count(*)
from ora.orders, ora.customer, ora.nation, ora.region
where o_custkey = c_custkey
and c_nationkey = n_nationkey
and n_regionkey = r_regionkey
and (char(year(o_orderdate))||char(month(o_orderdate))) >
((char(year(current date - 84 months))||char(month(current date - 84
months))))
group by r_name
;
```

2. Determine if the problem is at the federated server or the data source.

Once the problem query has been identified, we need to determine if the query's performance problem is at the federated server, the remote data source, or distributed between both.

This information can be determined from a dynamic SQL snapshot, as discussed in 4.2.3, "Federated server or remote data source" on page 136. Example 4-33 is a dynamic SQL snapshot, and includes the problem query (highlighted) in the Statement text field.

Example 4-33 Dynamic SQL snapshot

get snapshot for dynamic sql on fedserv

Dynamic SQL Snapshot Result

Database name	= FEDSERV
Database path	= /data1/npart/db2i64/NODE0000/SQL00001/
Number of executions	= 1
Number of compilations	= 1
Worst preparation time (ms)	= 57
Best preparation time (ms)	= 57
Internal rows deleted	= 0
Internal rows inserted	= 0
Rows read	= 5
Internal rows updated	= 0
Rows written	= 0
Statement sorts	= 1
Statement sort overflows	= 0
Total sort time	= 1098
Buffer pool data logical reads	= 4
Buffer pool data physical reads	= 1
Buffer pool temporary data logical reads	= 0
Buffer pool temporary data physical reads	= 0
Buffer pool index logical reads	= 6
Buffer pool index physical reads	= 3
Buffer pool temporary index logical reads	= 0
Buffer pool temporary index physical reads	= 0
Total execution time (sec.ms)	= 127.715346
Total user cpu time (sec.ms)	= 74.800000
Total system cpu time (sec.ms)	= 2.220000
Statement text	= select r_name, count(*) from ora.orders,
	ora.customer, ora.nation, ora.region where o_custkey = c_custkey and
	c_nationkey = n_nationkey and n_regionkey = r_regionkey and
	(char(year(o_orderdate)) char(month(o_orderdate))) > ((char(year(current date
	- 84 months)) char(month(current date - 84 months)))) group by r_name

Number of executions	= 1
Number of compilations	= 1
Worst preparation time (ms)	= 0
Best preparation time (ms)	= 0
Internal rows deleted	= 0
Internal rows inserted	= 0
Rows read	= 15000000
Internal rows updated	= 0
Rows written	= 0
Statement sorts	= 0
Buffer pool data logical reads	= 0
Buffer pool data physical reads	= 0
Buffer pool temporary data logical reads	= 0
Buffer pool temporary data physical reads	= 0
Buffer pool index logical reads	= 0
Buffer pool index physical reads	= 0
Buffer pool temporary index logical reads	= 0
Buffer pool temporary index physical reads	= 0
Total execution time (sec.ms)	= 87.370940
Total user cpu time (sec.ms)	= 0.000000
Total system cpu time (sec.ms)	= 0.000000
Statement text	= SELECT A0."O_ORDERDATE", A1."N_REGIONKEY"
FROM "IITEST"."ORDERS" A0, "IITEST"."NATION" A1, "IITEST"."CUSTOMER" A2 WHERE	
(A2."C_NATIONKEY" = A1."N_NATIONKEY") AND (A0."O_CUSTKEY" = A2."C_CUSTKEY")	

Number of executions	= 1
Number of compilations	= 1
Worst preparation time (ms)	= 0
Best preparation time (ms)	= 0
Internal rows deleted	= 0
Internal rows inserted	= 0
Rows read	= 5
Internal rows updated	= 0
Rows written	= 0
Statement sorts	= 0
Buffer pool data logical reads	= 0
Buffer pool data physical reads	= 0
Buffer pool temporary data logical reads	= 0
Buffer pool temporary data physical reads	= 0
Buffer pool index logical reads	= 0
Buffer pool index physical reads	= 0
Buffer pool temporary index logical reads	= 0
Buffer pool temporary index physical reads	= 0
Total execution time (sec.ms)	= 0.001234
Total user cpu time (sec.ms)	= 0.000000
Total system cpu time (sec.ms)	= 0.000000

Statement text "IITEST"."REGION" AO	= SELECT AO."R_REGIONKEY", AO."R_NAME" FROM
--	---

As discussed in 4.2.3, “Federated server or remote data source” on page 136, since there is no direct mechanism to link the remote SQL fragments in the dynamic cache with their corresponding user SQL statement, we generated **db2exfmt** output for the user query, as shown in Example 4-34, to obtain this link.

Example 4-34 db2exfmt output for the problem query

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
 Licensed Material - Program Property of IBM
 IBM DATABASE 2 Explain Table Format Tool

***** EXPLAIN INSTANCE *****

DB2_VERSION: 08.02.0
 SOURCE_NAME: SQLC2E05
 SOURCE_SCHEMA: NULLID
 SOURCE_VERSION:
 EXPLAIN_TIME: 2004-08-03-16.00.32.187732
 EXPLAIN_REQUESTER: DB2I64

Database Context:

 Parallelism: None
 CPU Speed: 5.392596e-07
 Comm Speed: 100
 Buffer Pool size: 75000
 Sort Heap size: 20000
 Database Heap size: 1200
 Lock List size: 100
 Maximum Lock List: 10
 Average Applications: 1
 Locks Available: 640

Package Context:

 SQL Type: Dynamic
 Optimization Level: 5
 Blocking: Block All Cursors
 Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 203 -----

QUERYNO: 1
 QUERYTAG:
 Statement Type: Select
 Updatable: No
 Deletable: No
 Query Degree: 1

Original Statement:

```

-----
select r_name, count(*)
from ora.orders, ora.customer, ora.nation, ora.region
where o_custkey = c_custkey and c_nationkey = n_nationkey and n_regionkey =
      r_regionkey and (char(year(o_orderdate))||char(month(o_orderdate))) >
      ((char(year(current date - 84 months))||(char(month(current date -
      84 months))))
group by r_name
  
```

Optimized Statement:

```

-----
SELECT Q6.$C0 AS "R_NAME", Q6.$C1
FROM
  (SELECT Q5.$C0, COUNT(* )
  FROM
    (SELECT Q1.R_NAME
    FROM ORA.REGION AS Q1, ORA.NATION AS Q2, ORA.CUSTOMER AS Q3, ORA.ORDERS
    AS Q4
    WHERE (Q2.N_REGIONKEY = Q1.R_REGIONKEY) AND (Q3.C_NATIONKEY =
    Q2.N_NATIONKEY) AND (Q4.O_CUSTKEY = Q3.C_CUSTKEY) AND
    ((CHAR(YEAR(-(CURRENT DATE, 84, 2))) || CHAR(MONTH(-(CURRENT
    DATE, 84, 2)))) < (CHAR(YEAR(Q4.O_ORDERDATE)) ||
    CHAR(MONTH(Q4.O_ORDERDATE)))))) AS Q5
  GROUP BY Q5.$C0) AS Q6
  
```

Access Plan:

 Total Cost: 1.02509e+06
 Query Degree:1

Rows
RETURN
(1)
Cost
I/O
5
GRPBY
(2)
1.02509e+06

```

220620
|
5
TBSCAN
( 3)
1.02509e+06
220620
|
5
SORT
( 4)
1.02509e+06
220620
|
7.50047e+06
HSJOIN
( 5)
1.02228e+06
220620
/-----+-----\
7.50047e+06 5
FILTER SHIP
( 6) ( 13)
1.02131e+06 0.00883415
220620 0
| |
2.25014e+07 5
SHIP NICKNM: ORA
( 7) REGION
951110
220620
+-----+-----+
25 1.5e+06 3e+07
NICKNM: ORA NICKNM: ORA NICKNM: ORA
NATION CUSTOMER ORDERS

```

1) RETURN: (Return Result)

Cumulative Total Cost: 1.02509e+06

Cumulative CPU Cost: 1.65932e+11

Cumulative I/O Cost: 220620

Cumulative Re-Total Cost: 1.02228e+06

Cumulative Re-CPU Cost: 1.60726e+11

Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 1.02509e+06

Estimated Bufferpool Buffers: 0

Remote communication cost:1.18169e+07

Arguments:

BLDLEVEL: (Build level)
DB2 v8.1.1.64 : s040509
ENVVAR : (Environment Variable)
DB2_EXTENDED_OPTIMIZATION = ON
STMTHEAP: (Statement heap size)
8192

Input Streams:

11) From Operator #2

Estimated number of rows: 5
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q7.\$C1+Q7.R_NAME

2) GRPBY : (Group By)
Cumulative Total Cost: 1.02509e+06
Cumulative CPU Cost: 1.65932e+11
Cumulative I/O Cost: 220620
Cumulative Re-Total Cost: 1.02228e+06
Cumulative Re-CPU Cost: 1.60726e+11
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 1.02509e+06
Estimated Bufferpool Buffers: 0
Remote communication cost:1.18169e+07

Arguments:

AGGMODE : (Aggregation Mode)
FINAL
GROUPBYC: (Group By columns)
TRUE
GROUPBYN: (Number of Group By columns)
1
GROUPBYR: (Group By requirement)
1: Q5.R_NAME
ONEFETCH: (One Fetch flag)
FALSE

Input Streams:

10) From Operator #3

Estimated number of rows: 5
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q5.R_NAME(A)

Output Streams:

11) To Operator #1

Estimated number of rows: 5
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q7.\$C1+Q7.R_NAME

3) TBSCAN: (Table Scan)

Cumulative Total Cost: 1.02509e+06
Cumulative CPU Cost: 1.65932e+11
Cumulative I/O Cost: 220620
Cumulative Re-Total Cost: 1.02228e+06
Cumulative Re-CPU Cost: 1.60726e+11
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 1.02509e+06
Estimated Bufferpool Buffers: 0
Remote communication cost:1.18169e+07

Arguments:

MAXPAGES: (Maximum pages for prefetch)
ALL
PREFETCH: (Type of Prefetch)
NONE
SCANDIR : (Scan Direction)
FORWARD

Input Streams:

9) From Operator #4

Estimated number of rows: 5

Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q5.R_NAME(A)

Output Streams:

10) To Operator #2

Estimated number of rows: 5
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q5.R_NAME(A)

4) SORT : (Sort)

Cumulative Total Cost: 1.02509e+06
Cumulative CPU Cost: 1.65932e+11
Cumulative I/O Cost: 220620
Cumulative Re-Total Cost: 1.02228e+06
Cumulative Re-CPU Cost: 1.60726e+11
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 1.02509e+06
Estimated Bufferpool Buffers: 220622
Remote communication cost:1.18169e+07

Arguments:

AGGMODE : (Aggregation Mode)
PARTIAL
DUPLWARN: (Duplicates Warning flag)
FALSE
NUMROWS : (Estimated number of rows)
5
ROWWIDTH: (Estimated width of rows)
36
SORTKEY : (Sort Key column)
1: Q5.R_NAME(A)
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

8) From Operator #5

Estimated number of rows: 7.50047e+06
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q5.R_NAME

Output Streams:

9) To Operator #3

Estimated number of rows: 5
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q5.R_NAME(A)

5) HSJOIN: (Hash Join)

Cumulative Total Cost: 1.02228e+06
Cumulative CPU Cost: 1.60726e+11
Cumulative I/O Cost: 220620
Cumulative Re-Total Cost: 1.02228e+06
Cumulative Re-CPU Cost: 1.60726e+11
Cumulative Re-I/O Cost: 220620
Cumulative First Row Cost: 1.02228e+06
Estimated Bufferpool Buffers: 220622
Remote communication cost:1.18169e+07

Arguments:

BITFLTR : (Hash Join Bit Filter used)
FALSE
EARLYOUT: (Early Out flag)
NONE
HASHCODE: (Hash Code Size)
24 BIT
TEMPSIZE: (Temporary Table Page Size)
4096

Predicates:

3) Predicate used in Join
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 0.2

Predicate Text:

(Q2.N_REGIONKEY = Q1.R_REGIONKEY)

Input Streams:

5) From Operator #6

Estimated number of rows: 7.50047e+06
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_REGIONKEY+Q4.O_ORDERDATE

7) From Operator #13

Estimated number of rows: 5
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q1.R_NAME+Q1.R_REGIONKEY

Output Streams:

8) To Operator #4

Estimated number of rows: 7.50047e+06
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q5.R_NAME

6) FILTER: (Filter)
Cumulative Total Cost: 1.02131e+06

Cumulative CPU Cost: 1.58917e+11
Cumulative I/O Cost: 220620
Cumulative Re-Total Cost: 1.02131e+06
Cumulative Re-CPU Cost: 1.58917e+11
Cumulative Re-I/O Cost: 220620
Cumulative First Row Cost: 951110
Estimated Bufferpool Buffers: 220621
Remote communication cost:1.18169e+07

Arguments:

JN INPUT: (Join input leg)
OUTER

Predicates:

6) Residual Predicate
Relational Operator: Less Than (<)
Subquery Input Required: No
Filter Factor: 0.333333

Predicate Text:

((CHAR(YEAR(-(CURRENT DATE, 84, 2))) ||
CHAR(MONTH(-(CURRENT DATE, 84, 2)))) <
(CHAR(YEAR(Q4.O_ORDERDATE)) ||
CHAR(MONTH(Q4.O_ORDERDATE))))

Input Streams:

4) From Operator #7

Estimated number of rows: 2.25014e+07
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_REGIONKEY+Q4.O_ORDERDATE

Output Streams:

5) To Operator #5

Estimated number of rows: 7.50047e+06
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_REGIONKEY+Q4.O_ORDERDATE

7) SHIP : (Ship)

Cumulative Total Cost: 951110
Cumulative CPU Cost: 2.87461e+10
Cumulative I/O Cost: 220620
Cumulative Re-Total Cost: 951110
Cumulative Re-CPU Cost: 2.87461e+10
Cumulative Re-I/O Cost: 220620
Cumulative First Row Cost: 951110
Estimated Bufferpool Buffers: 220621
Remote communication cost:1.18169e+07

Arguments:

CSERQY : (Remote common subexpression)

FALSE

DSTSEVER: (Destination (ship to) server)

- (NULL).

RMTQTX : (Remote statement)

SELECT A0."O_ORDERDATE", A1."N_REGIONKEY" FROM "IITEST"."ORDERS" A0,
"IITEST"."NATION" A1, "IITEST"."CUSTOMER" A2 WHERE (A2."C_NATIONKEY" =
A1."N_NATIONKEY") AND (A0."O_CUSTKEY" = A2."C_CUSTKEY")

SRCSEVER: (Source (ship from) server)

ORASERV

STREAM : (Remote stream)

FALSE

Input Streams:

1) From Object ORA.ORDERS

Estimated number of rows: 3e+07

Number of columns: 3

Subquery predicate ID: Not Applicable

Column Names:

+Q4.\$RID\$+Q4.O_ORDERDATE+Q4.O_CUSTKEY

2) From Object ORA.NATION

Estimated number of rows: 25

Number of columns: 3

Subquery predicate ID: Not Applicable

```

Column Names:
-----
+Q2.$RID$+Q2.N_REGIONKEY+Q2.N_NATIONKEY

```

3) From Object ORA.CUSTOMER

```

Estimated number of rows: 1.5e+06
Number of columns: 3
Subquery predicate ID: Not Applicable

```

```

Column Names:
-----
+Q3.$RID$+Q3.C_NATIONKEY+Q3.C_CUSTKEY

```

Output Streams:

4) To Operator #6

```

Estimated number of rows: 2.25014e+07
Number of columns: 2
Subquery predicate ID: Not Applicable

```

```

Column Names:
-----
+Q2.N_REGIONKEY+Q4.O_ORDERDATE

```

13) SHIP : (Ship)

```

Cumulative Total Cost: 0.00883415
Cumulative CPU Cost: 16382
Cumulative I/O Cost: 0
Cumulative Re-Total Cost: 0.00171431
Cumulative Re-CPU Cost: 3179
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 0.0028295
Estimated Bufferpool Buffers: 1
Remote communication cost:11.3594

```

Arguments:

```

-----
CSERQY : (Remote common subexpression)
        FALSE
DSTSEVER: (Destination (ship to) server)
        - (NULL).
JN INPUT: (Join input leg)
        INNER
RMTQTX : (Remote statement)

```

```
SELECT AO."R_REGIONKEY", AO."R_NAME" FROM "IITEST"."REGION" AO
SRCSEVER: (Source (ship from) server)
ORASERV
STREAM : (Remote stream)
FALSE
```

Input Streams:

```
-----
6) From Object ORA.REGION

Estimated number of rows: 5
Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:
-----
+Q1.$RID$+Q1.R_NAME+Q1.R_REGIONKEY
```

Output Streams:

```
-----
7) To Operator #5

Estimated number of rows: 5
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:
-----
+Q1.R_NAME+Q1.R_REGIONKEY
```

Objects Used in Access Plan:

```
-----
Schema: ORA
Name: CUSTOMER
Type: Nickname
Time of creation: 2004-08-03-15.26.18.435035
Last statistics update:
Number of columns: 8
Number of rows: 1500000
Width of rows: 42
Number of buffer pool pages: 32173
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
```

Prefetch page count: 32
Container extent page count: 32

Schema: ORA
Name: NATION
Type: Nickname
Time of creation: 2004-06-16-23.22.43.109494
Last statistics update:
Number of columns: 4
Number of rows: 25
Width of rows: 42
Number of buffer pool pages: 15
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: ORA
Name: ORDERS
Type: Nickname
Time of creation: 2004-06-18-18.14.00.595431
Last statistics update: 2004-06-21-09.48.39.550659
Number of columns: 9
Number of rows: 30000000
Width of rows: 41
Number of buffer pool pages: 217186
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: ORA
Name: REGION
Type: Nickname
Time of creation: 2004-06-16-23.22.43.342734
Last statistics update:
Number of columns: 3
Number of rows: 5
Width of rows: 56
Number of buffer pool pages: 15
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000

Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

The RMTQTX field of the SHIP operator 7 in Example 4-34 on page 214 contains the remote SQL fragment:

```
SELECT A0."O_ORDERDATE", A1."N_REGIONKEY" FROM "IITEST"."ORDERS" A0,  
"IITEST"."NATION" A1, "IITEST"."CUSTOMER" A2 WHERE (A2."C_NATIONKEY" =  
A1."N_NATIONKEY") AND (A0."O_CUSTKEY" = A2."C_CUSTKEY")
```

The RMTQTX field of the SHIP operator 13 in Example 4-34 on page 214 contains the remote SQL fragment:

```
SELECT A0."R_REGIONKEY", A0."R_NAME" FROM "IITEST"."REGION" A0
```

These remote SQL fragments are located in the dynamic cache output shown in Example 4-33 on page 212, and the following information can be gathered:

- ▶ **Number of executions** is 1 for the user-entered query as well as the two remote SQL fragments.
- ▶ **Total execution time (sec.ms)**, which is 127.715346 seconds for the user-entered query, and 87.370940 seconds and 0.001234 seconds for each of the remote SQL fragments.
- ▶ **Total user cpu time (sec.ms)** and **Total system cpu time (sec.ms)**, which is 74.800000 and 2.220000 seconds for the user-entered query, and zero for the two remote SQL fragments.
- ▶ **Rows read** is 5 for the user-entered query, which is the number of rows returned to the user, and 15000000 rows and 5 rows from each of the remote SQL fragment data sources returned to the federated server.
- ▶ Other fields of interest include **Statement sorts**, **Statement sort overflows**, and **Total sort time**, which only apply to the user-entered query, and have values 1, zero, and 1098 milliseconds, respectively.

Note: To obtain the average elapsed and CPU times, as well as the number of rows returned, you must divide the values shown by the **Number of executions**.

We can derive the following information from these metrics:

- ▶ The average number of rows returned from each remote data source to the federated server is $(15000000 / 1) = 15000000$ and $(5 / 1) = 5$ rows, respectively.

- The average elapsed time for the user query is $(127.715346 / 1) = 127.715346$ seconds, while that of each remote SQL fragment is $(87.370940 / 1) = 87.370940$ seconds and $(0.001234 / 1) = 0.001234$ seconds, respectively.

Attention: In our example, given that the total query elapsed time is 127.715346 seconds, it is clear that more than two thirds of the elapsed time of the query is spent at the remote data source. $((87.370940 + 0.001234) / 127.715346) \times 100 = 68.41\%$ of the query elapsed time is accounted for at the remote data source indicating that to be the potential source of the performance problem.

Since the predominant processing time is consumed at one of the remote data sources (ORDERS, NATION, and CUSTOMER tables), we need to investigate remote data source related items, as described in 4.2.5, “Remote data source related” on page 161.

The **Rows read** field in the dynamic cache output for one of the remote SQL fragments in Example 4-33 on page 212 has 15000000 rows being returned to the federated server, while the SHIP operator 7 in the Access Plan section of Example 4-34 on page 214 estimates that $2.25014e+07$ rows would be returned to the federated server, and the SHIP operator 13 in the Access Plan section of Example 4-34 on page 214 estimates that 5 rows would be returned to the federated server. The various Objects used in the Access Plan section of Example 4-34 on page 214 shows the Number of rows being 5 for ORA.REGION, 25 for ORA.NATION, $1.5e+06$ for the ORA.CUSTOMER, and $3e+07$ for ORA.ORDERS. nickname. This information is obtained from nickname statistics recorded in the global catalog.

Note: At this point, we cross checked this explain output with a history of explain output for this query, and determined that previous explains had exploited an aggregate MQT that had been created on these four nicknames to improve the performance of precisely such queries. The **db2exfmt** output in Example 4-34 on page 214 shows no MQT being used.

We therefore decided to investigate MQTs dependent on these nicknames, and determined (using SQL against the catalog—query not shown here) that there were *no* MQTs dependent on these four nicknames.

Root cause of the problem

The root cause of the problem was that an MQT that had been designed specifically for this query had somehow been dropped, resulting in poor performance times for the query.

A further investigation indicated that the ORA.CUSTOMER and ORA.ORDERS nicknames had been dropped and recreated to resynchronize the index information and the statistics, which had resulted in all MQTs dependent on these nicknames to be dropped. Human error had caused the dependent MQTs not to be recreated after the nicknames had been recreated.

Note: We strongly recommend that the Statistics Update facility or **NNSTAT** stored procedure and manual procedures be used to synchronize the global catalog with the remote data source to avoid precisely such undesirable side effects.

Apply best practices

In this case, the solution involves recreating the MQT, verifying (using **db2exfmt**) that the DB2 optimizer had indeed routed the query to the ORDERSMQT, and then executing the query.

The ORDERSMQT creation, **db2exfmt** output, and dynamic cache snapshot are shown in Example 4-35, Example 4-36 on page 230, and Example 4-37 on page 238, respectively.

Example 4-35 MQT definition for ORDERSMQT

connect to fedserv

Database Connection Information

Database server = DB2/AIX64 8.2.0
SQL authorization ID = DB2I64
Local database alias = FEDSERV

```
CREATE TABLE ORDERSMQT (REGION, YRMONTH, COUNT)
AS ( SELECT r_name,
char(year(o_orderdate))||char(month(o_orderdate)) as yrmonth ,
count(*)
from ora.orders, ora.customer, ora.nation, ora.region
where o_custkey = c_custkey
and c_nationkey = n_nationkey
and n_regionkey = r_regionkey
group by r_name, char(year(o_orderdate))||char(month(o_orderdate)) )
DATA INITIALLY DEFERRED
REFRESH DEFERRED
DB20000I The SQL command completed successfully.
```

```
REFRESH TABLE ORDERSMQT
DB20000I The SQL command completed successfully.
```

```
SET CURRENT REFRESH AGE ANY
DB20000I The SQL command completed successfully.
```

Example 4-36 db2exfmt output with MQT

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

***** EXPLAIN INSTANCE *****

DB2_VERSION: 08.02.0
SOURCE_NAME: SQLC2E05
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-07-07-09.47.54.023210
EXPLAIN_REQUESTER: DB2I64

Database Context:

Parallelism: None
CPU Speed: 5.392596e-07
Comm Speed: 100
Buffer Pool size: 75000
Sort Heap size: 20000
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 640

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 203 -----

QUERYNO: 1
QUERYTAG:
Statement Type: Select
Updatable: No

Deletable: No
Query Degree: 1

Original Statement:

```
-----  
select r_name, count(*)  
from ora.orders, ora.customer, ora.nation, ora.region  
where o_custkey = c_custkey and c_nationkey = n_nationkey and n_regionkey =  
      r_regionkey and (char(year(o_orderdate))||char(month(o_orderdate))) >  
      ((char(year(current date - 74 months))|| (char(month(current date -74  
      months))))  
group by r_name
```

Optimized Statement:

```
-----  
SELECT Q3.$C0 AS "R_NAME", Q3.$C1  
FROM  
  (SELECT Q2.$C0, SUM(Q2.$C1)  
   FROM  
     (SELECT Q1.REGION, Q1.COUNT  
      FROM DB2I64.ORDERSMQT AS Q1  
      WHERE ((CHAR(YEAR(-(CURRENT DATE, 74, 2))) || CHAR(MONTH(-(CURRENT DATE,  
      74, 2)))) < Q1.YRMONTH)) AS Q2  
   GROUP BY Q2.$C0) AS Q3
```

Access Plan:

```
-----  
Total Cost: 176.091  
Query Degree:1
```

```
Rows  
RETURN  
( 1)  
Cost  
I/O  
|  
14.8333  
GRPBY  
( 2)  
176.088  
7  
|  
14.8333  
TBSCAN  
( 3)  
176.085  
7  
|
```

```

14.8333
SORT
( 4)
176.076
  7
  |
148.333
TBSCAN
( 5)
175.998
  7
  |
  445
TABLE: DB2I64
ORDERSMQT

```

1) RETURN: (Return Result)

Cumulative Total Cost: 176.091
 Cumulative CPU Cost: 2.02393e+06
 Cumulative I/O Cost: 7
 Cumulative Re-Total Cost: 0.963306
 Cumulative Re-CPU Cost: 1.78635e+06
 Cumulative Re-I/O Cost: 0
 Cumulative First Row Cost: 176.077
 Estimated Bufferpool Buffers: 0

Arguments:

```

-----
BLDLEVEL: (Build level)
          DB2 v8.1.1.64 : s040509
ENVVAR   : (Environment Variable)
          DB2_EXTENDED_OPTIMIZATION = ON
STMHEAP: (Statement heap size)
          8192

```

Input Streams:

5) From Operator #2

Estimated number of rows: 14.8333
 Number of columns: 2
 Subquery predicate ID: Not Applicable

Column Names:

```

-----
+Q4.$C1+Q4.R_NAME

```

```
2) GRPBY : (Group By)
Cumulative Total Cost: 176.088
Cumulative CPU Cost: 2.01688e+06
Cumulative I/O Cost: 7
Cumulative Re-Total Cost: 0.959506
Cumulative Re-CPU Cost: 1.7793e+06
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 176.077
Estimated Bufferpool Buffers: 0
```

Arguments:

```
-----
AGGMODE : (Aggregation Mode)
        FINAL
GROUPBYC: (Group By columns)
        TRUE
GROUPBYN: (Number of Group By columns)
        1
GROUPBYR: (Group By requirement)
        1: Q2.REGION
ONEFETCH: (One Fetch flag)
        FALSE
```

Input Streams:

```
-----
4) From Operator #3
```

```
Estimated number of rows: 14.8333
Number of columns: 2
Subquery predicate ID: Not Applicable
```

Column Names:

```
-----
+Q2.REGION(A)+Q2.COUNT
```

Output Streams:

```
-----
5) To Operator #1
```

```
Estimated number of rows: 14.8333
Number of columns: 2
Subquery predicate ID: Not Applicable
```

Column Names:

```
-----
+Q4.$C1+Q4.R_NAME
```

3) TBSCAN: (Table Scan)
 Cumulative Total Cost: 176.085
 Cumulative CPU Cost: 2.01292e+06
 Cumulative I/O Cost: 7
 Cumulative Re-Total Cost: 0.957372
 Cumulative Re-CPU Cost: 1.77534e+06
 Cumulative Re-I/O Cost: 0
 Cumulative First Row Cost: 176.077
 Estimated Bufferpool Buffers: 0

Arguments:

 MAXPAGES: (Maximum pages for prefetch)
 ALL
 PREFETCH: (Type of Prefetch)
 NONE
 SCANDIR : (Scan Direction)
 FORWARD

Input Streams:

 3) From Operator #4

Estimated number of rows: 14.8333
 Number of columns: 2
 Subquery predicate ID: Not Applicable

Column Names:

 +Q2.REGION(A)+Q2.COUNT

Output Streams:

 4) To Operator #2

Estimated number of rows: 14.8333
 Number of columns: 2
 Subquery predicate ID: Not Applicable

Column Names:

 +Q2.REGION(A)+Q2.COUNT

4) SORT : (Sort)
 Cumulative Total Cost: 176.076

Cumulative CPU Cost: 1.99536e+06
Cumulative I/O Cost: 7
Cumulative Re-Total Cost: 0.947902
Cumulative Re-CPU Cost: 1.75778e+06
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 176.076
Estimated Bufferpool Buffers: 7

Arguments:

AGGMODE : (Aggregation Mode)
PARTIAL
DUPLWARN: (Duplicates Warning flag)
FALSE
NUMROWS : (Estimated number of rows)
15
ROWWIDTH: (Estimated width of rows)
41
SORTKEY : (Sort Key column)
1: Q2.REGION(A)
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

2) From Operator #5

Estimated number of rows: 148.333
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.COUNT+Q2.REGION

Output Streams:

3) To Operator #3

Estimated number of rows: 14.8333
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.REGION(A)+Q2.COUNT

```

5) TBSCAN: (Table Scan)
Cumulative Total Cost: 175.998
Cumulative CPU Cost: 1.85042e+06
Cumulative I/O Cost: 7
Cumulative Re-Total Cost: 0.947902
Cumulative Re-CPU Cost: 1.75778e+06
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 25.6885
Estimated Bufferpool Buffers: 7

Arguments:
-----
MAXPAGES: (Maximum pages for prefetch)
ALL
PREFETCH: (Type of Prefetch)
NONE
ROWLOCK : (Row Lock intent)
NEXT KEY SHARE
SCANDIR : (Scan Direction)
FORWARD
TABLOCK : (Table Lock intent)
INTENT SHARE
TBISOLVL: (Table access Isolation Level)
CURSOR STABILITY

Predicates:
-----
3) Sargable Predicate
Relational Operator: Less Than (<)
Subquery Input Required: No
Filter Factor: 0.333333

Predicate Text:
-----
( (CHAR(YEAR(-(CURRENT DATE, 74, 2))) ||
CHAR(MONTH(-(CURRENT DATE, 74, 2)))) <
Q1.YRMONTH)

Input Streams:
-----
1) From Object DB2I64.ORDERSMQT

Estimated number of rows: 445
Number of columns: 4
Subquery predicate ID: Not Applicable

```

Column Names:

+Q1.\$RID\$+Q1.COUNT+Q1.REGION+Q1.YRMONTH

Output Streams:

2) To Operator #4

Estimated number of rows: 148.333
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.COUNT+Q2.REGION

Objects Used in Access Plan:

Schema: ORA
Name: CUSTOMER
Type: Nickname (reference only)

Schema: ORA
Name: NATION
Type: Nickname (reference only)

Schema: ORA
Name: ORDERS
Type: Nickname (reference only)

Schema: ORA
Name: REGION
Type: Nickname (reference only)

Schema: DB2I64
Name: ORDERSMQT
Type: Table
Time of creation: 2004-07-06-15.53.07.922469
Last statistics update:
Number of columns: 3
Number of rows: 445
Width of rows: 57
Number of buffer pool pages: 7
Distinct row values: No
Tablespace name: INDEX_TS
Tablespace overhead: 24.100000

Tablespace transfer rate: 0.900000
 Source for statistics: Single Node
 Prefetch page count: 96
 Container extent page count: 32
 Table overflow record count: 0
 Table Active Blocks: -1

Example 4-37 Dynamic SQL snapshot with ORDERSMQT

Dynamic SQL Snapshot Result

Database name	= FEDSERV
Database path	= /data1/npart/db2i64/NODE0000/SQL00001/
Number of executions	= 1
Number of compilations	= 1
Worst preparation time (ms)	= 28
Best preparation time (ms)	= 28
Internal rows deleted	= 0
Internal rows inserted	= 0
Rows read	= 401
Internal rows updated	= 0
Rows written	= 0
Statement sorts	= 1
Statement sort overflows	= 0
Total sort time	= 13
Buffer pool data logical reads	= 9
Buffer pool data physical reads	= 6
Buffer pool temporary data logical reads	= 0
Buffer pool temporary data physical reads	= 0
Buffer pool index logical reads	= 0
Buffer pool index physical reads	= 0
Buffer pool temporary index logical reads	= 0
Buffer pool temporary index physical reads	= 0
Total execution time (sec.ms)	= 0.048358
Total user cpu time (sec.ms)	= 0.000000
Total system cpu time (sec.ms)	= 0.010000
Statement text	= select r_name, count(*) from ora.orders, ora.customer, ora.nation, ora.region where o_custkey = c_custkey and c_nationkey = n_nationkey and n_regionkey = r_regionkey and (char(year(o_orderdate)) char(month(o_orderdate))) > ((char(year(current date - 74 months)) char(month(current date -74 months)))) group by r_name

Example 4-36 on page 230 shows the query now using ORDERSMQT instead of the four nicknames. Example 4-37 on page 238 shows the results of executing the query with routing to the ORDERSMQT in effect.

Note: There is no remote SQL fragment in this output because of the routing, and because the **Total execution time (sec.ms)** went from 128.136489 seconds in Example 4-33 on page 212 without routing to 0.048358 seconds with the routing to ORDERSMQT.

4.4.5 Incompatible data types on join columns

Table 4-2 on page 158 provides a very high-level overview of conditions under which one of the three join strategies (nested loop, merge scan, or hash join) is favored over the other. A key consideration is the join operator involved and whether the data types, scale, and precision of the joined columns match perfectly. For example, a join operator other than equality can only result in a nested loop join, while equality predicates are a must for merge scan and hash join. Additionally, hash join requires the join columns to match perfectly on data type, scale, and precision.

In a federated environment, joins can occur between nicknames referencing different data sources, and the likelihood of data type mismatches between joined columns is higher because of the default data type mapping that occurs when a nickname is defined. This can result in hash joins being inhibited altogether even though it might have been the optimal access paths had the joined columns matched perfectly.

In this scenario, we highlight such a mismatch that may be the cause of the performance problem, and describe the steps involved to correct this mismatch by altering the data type of the joined column.

Triggering event

Users complained about poor response times with a specific query. The users claimed that they had never really experienced good performance from this query.

Hypotheses and validations

Here too, since these were user complaints about the performance of a specific query, as per Figure 4-3 on page 122, we decided to enter the DB2 hypotheses hierarchy described in Figure 4-1 on page 117 and Example 4.2 on page 119, at a lower level, bypassing network- and system-related problems, and focused directly on federated application/query performance as follows.

Hypothesis 1: Federated application/query performance

Before one can investigate the cause of a query’s performance problem, one needs to identify the query in question. After identifying the query in question, one can begin diagnosing whether the performance problem is at the federated server, the remote data source, or equally divided between the two, as discussed in Example 4.2 on page 119.

- Identify the application and query.

In this case the user specifically identified the query in question as being the one shown in Example 4-38. It lists all orders (DB2.ORDERS) and the line items (ORA.LINEITEM) associated wit them.

Example 4-38 Problem query

```
select * from db2.orders d, ora.lineitem l
where d.o_orderkey = l.l_orderkey
```

- Determine if the problem is at the federated server or the data source.

Once the problem query has been identified, we need to determine if the query’s performance problem is at the federated server, the remote data source, or distributed between both.

This information can be determined from a dynamic SQL snapshot, as discussed in 4.2.3, “Federated server or remote data source” on page 136. Example 4-39 on page 240 is a dynamic SQL snapshot, and includes the problem query (highlighted) in the Statement text field.

Example 4-39 Dynamic SQL snapshot

```
get snapshot for dynamic sql on fedserv
```

Dynamic SQL Snapshot Result

Database name	= FEDSERV
Database path	= /data1/npart/db2i64/NODE0000/SQL00001/
Number of executions	= 1
Number of compilations	= 1
Worst preparation time (ms)	= 48
Best preparation time (ms)	= 48
Internal rows deleted	= 0
Internal rows inserted	= 0
Rows read	= 59986052
Internal rows updated	= 0
Rows written	= 64157889
Statement sorts	= 2

Statement sort overflows = 2
Total sort time = 479027
 Buffer pool data logical reads = 6
 Buffer pool data physical reads = 2
Buffer pool temporary data logical reads = 5653175
Buffer pool temporary data physical reads = 184
 Buffer pool index logical reads = 7
 Buffer pool index physical reads = 4
 Buffer pool temporary index logical reads = 0
 Buffer pool temporary index physical reads = 0
Total execution time (sec.ms) = 3338.542900
Total user cpu time (sec.ms) = 1070.300000
Total system cpu time (sec.ms) = 58.810000
Statement text = select * from db2.orders d, ora.lineitem
 l where d.o_orderkey = l.l_orderkey

Number of executions = 1
 Number of compilations = 1
 Worst preparation time (ms) = 0
 Best preparation time (ms) = 0
 Internal rows deleted = 0
 Internal rows inserted = 0
Rows read = 59986052
 Internal rows updated = 0
 Rows written = 0
 Statement sorts = 0
 Buffer pool data logical reads = 0
 Buffer pool data physical reads = 0
 Buffer pool temporary data logical reads = 0
 Buffer pool temporary data physical reads = 0
 Buffer pool index logical reads = 0
 Buffer pool index physical reads = 0
 Buffer pool temporary index logical reads = 0
 Buffer pool temporary index physical reads = 0
Total execution time (sec.ms) = 1870.848680
 Total user cpu time (sec.ms) = 0.000000
 Total system cpu time (sec.ms) = 0.000000
Statement text = SELECT A0."L_ORDERKEY", A0."L_PARTKEY",
 A0."L_SUPPKEY", A0."L_LINENUMBER", A0."L_QUANTITY", A0."L_EXTENDEDPRICE",
 A0."L_DISCOUNT",
 A0."L_TAX", A0."L_RETURNFLAG", A0."L_LINESTATUS", A0."L_SHIPDATE",
 A0."L_COMMITDATE",
 A0."L_RECEIPTDATE", A0."L_SHIPINSTRUCT", A0."L_SHIPMODE", A0."L_COMMENT"
 FROM "IITEST"."LINEITEM" A0

Number of executions = 1
 Number of compilations = 1

```

Worst preparation time (ms)      = 0
Best preparation time (ms)      = 0
Internal rows deleted           = 0
Internal rows inserted          = 0
Rows read                      = 15000000
Internal rows updated           = 0
Rows written                    = 0
Statement sorts                 = 0
Buffer pool data logical reads  = 0
Buffer pool data physical reads = 0
Buffer pool temporary data logical reads = 0
Buffer pool temporary data physical reads = 0
Buffer pool index logical reads = 0
Buffer pool index physical reads = 0
Buffer pool temporary index logical reads = 0
Buffer pool temporary index physical reads = 0
Total execution time (sec.ms) = 333.479573
Total user cpu time (sec.ms)    = 0.000000
Total system cpu time (sec.ms)  = 0.000000
Statement text                = SELECT AO."O_ORDERKEY", AO."O_CUSTKEY",
AO."O_ORDERSTATUS", AO."O_TOTALPRICE", AO."O_ORDERDATE", AO."O_ORDERPRIORITY",
AO."O_CLERK", AO."O_SHIPPRIORITY", AO."O_COMMENT" FROM "TPCD"."ORDERS" AO FOR
READ ONLY

```

As discussed in 4.2.3, “Federated server or remote data source” on page 136, since there is no direct mechanism to link the remote SQL fragments in the dynamic cache with their corresponding user SQL statement, we generated **db2exfmt** output for the user query, as shown in Example 4-40, to obtain this link.

Example 4-40 db2exfmt output for the problem query

```

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

```

```

***** EXPLAIN INSTANCE *****

```

```

DB2_VERSION: 08.02.0
SOURCE_NAME: SQLC2E05
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-08-05-17.38.11.227639
EXPLAIN_REQUESTER: DB2I64

```

```

Database Context:
-----

```


Parallelism: None
CPU Speed: 5.392596e-07
Comm Speed: 100
Buffer Pool size: 75000
Sort Heap size: 100000
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 640

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 203 -----

QUERYNO: 1
QUERYTAG:
Statement Type: Select
Udatable: No
Deletable: No
Query Degree: 1

Original Statement:

select *
from db2.orders d, ora.lineitem l
where d.o_orderkey = l.l_orderkey

Optimized Statement:

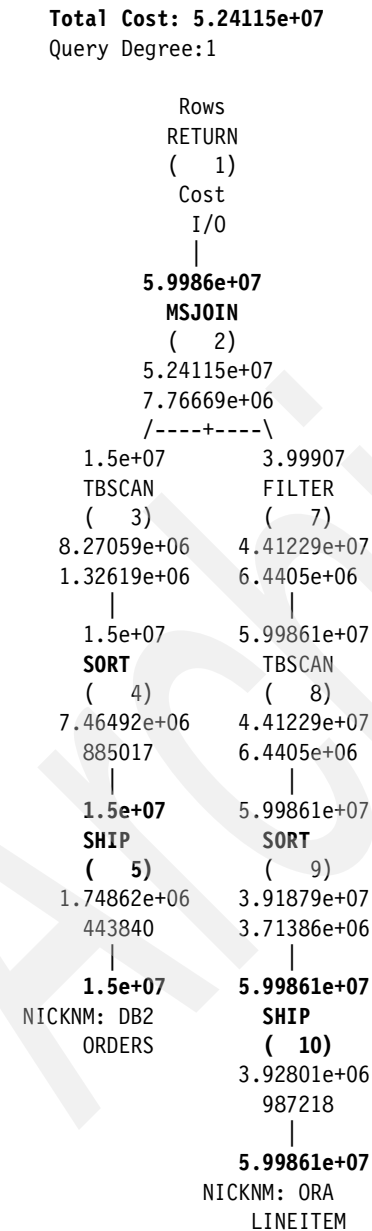
SELECT Q2.O_ORDERKEY AS "O_ORDERKEY", Q2.O_CUSTKEY AS "O_CUSTKEY",
Q2.O_ORDERSTATUS AS "O_ORDERSTATUS", Q2.O_TOTALPRICE AS
"O_TOTALPRICE", Q2.O_ORDERDATE AS "O_ORDERDATE", Q2.O_ORDERPRIORITY
AS "O_ORDERPRIORITY", Q2.O_CLERK AS "O_CLERK", Q2.O_SHIPPRIORITY AS
"O_SHIPPRIORITY", Q2.O_COMMENT AS "O_COMMENT", Q1.L_ORDERKEY AS
"L_ORDERKEY", Q1.L_PARTKEY AS "L_PARTKEY", Q1.L_SUPPKEY AS
"L_SUPPKEY", Q1.L_LINENUMBER AS "L_LINENUMBER", Q1.L_QUANTITY AS
"L_QUANTITY", Q1.L_EXTENDEDPRICE AS "L_EXTENDEDPRICE", Q1.L_DISCOUNT
AS "L_DISCOUNT", Q1.L_TAX AS "L_TAX", Q1.L_RETURNFLAG AS
"L_RETURNFLAG", Q1.L_LINESTATUS AS "L_LINESTATUS", Q1.L_SHIPDATE AS
"L_SHIPDATE", Q1.L_COMMITDATE AS "L_COMMITDATE", Q1.L_RECEIPTDATE AS
"L_RECEIPTDATE", Q1.L_SHIPINSTRUCT AS "L_SHIPINSTRUCT", Q1.L_SHIPMODE

```

        AS "L_SHIPMODE", Q1.L_COMMENT AS "L_COMMENT"
FROM ORA.LINEITEM AS Q1, DB2.ORDERS AS Q2
WHERE (Q2.O_ORDERKEY = Q1.L_ORDERKEY)

```

Access Plan:



1) RETURN: (Return Result)
 Cumulative Total Cost: 5.24115e+07
 Cumulative CPU Cost: 9.61945e+11
 Cumulative I/O Cost: 7.76669e+06
 Cumulative Re-Total Cost: 5.24115e+07
 Cumulative Re-CPU Cost: 9.61945e+11
 Cumulative Re-I/O Cost: 7.76669e+06
 Cumulative First Row Cost: 4.67981e+07
 Estimated Bufferpool Buffers: 3.16782e+06
 Remote communication cost:5.29733e+07

Arguments:

 BLDLEVEL: (Build level)
 DB2 v8.1.1.64 : s040509
 ENVVAR : (Environment Variable)
 DB2_EXTENDED_OPTIMIZATION = ON
 STMTHEAP: (Statement heap size)
 8192

Input Streams:

 10) From Operator #2

Estimated number of rows: 5.9986e+07
 Number of columns: 25
 Subquery predicate ID: Not Applicable

Column Names:

 +Q3.L_COMMENT+Q3.L_SHIPMODE+Q3.L_SHIPINSTRUCT
 +Q3.L_RECEIPTDATE+Q3.L_COMMITDATE
 +Q3.L_SHIPDATE+Q3.L_LINESTATUS+Q3.L_RETURNFLAG
 +Q3.L_TAX+Q3.L_DISCOUNT+Q3.L_EXTENDEDPRICE
 +Q3.L_QUANTITY+Q3.L_LINENUMBER+Q3.L_SUPPKEY
 +Q3.L_PARTKEY+Q3.L_ORDERKEY+Q3.O_COMMENT
 +Q3.O_SHIPPRIORITY+Q3.O_CLERK
 +Q3.O_ORDERPRIORITY+Q3.O_ORDERDATE
 +Q3.O_TOTALPRICE+Q3.O_ORDERSTATUS+Q3.O_CUSTKEY
 +Q3.O_ORDERKEY

2) MSJOIN: (Merge Scan Join)
 Cumulative Total Cost: 5.24115e+07
 Cumulative CPU Cost: 9.61945e+11
 Cumulative I/O Cost: 7.76669e+06

Cumulative Re-Total Cost: 5.24115e+07
Cumulative Re-CPU Cost: 9.61945e+11
Cumulative Re-I/O Cost: 7.76669e+06
Cumulative First Row Cost: 4.67981e+07
Estimated Bufferpool Buffers: 3.16782e+06
Remote communication cost:5.29733e+07

Arguments:

EARLYOUT: (Early Out flag)
 NONE
INNERCOL: (Inner Order Columns)
 1: Q1.L_ORDERKEY(A)
OUTERCOL: (Outer Order columns)
 1: Q2.0_ORDERKEY(A)
TEMPSIZE: (Temporary Table Page Size)
 4096

Predicates:

2) Predicate used in Join
 Relational Operator: Equal (=)
 Subquery Input Required: No
 Filter Factor: 6.66667e-08

Predicate Text:

(Q2.0_ORDERKEY = Q1.L_ORDERKEY)

Input Streams:

 4) From Operator #3

 Estimated number of rows: 1.5e+07
 Number of columns: 9
 Subquery predicate ID: Not Applicable

 Column Names:

 +Q2.0_ORDERKEY(A)+Q2.0_COMMENT
 +Q2.0_SHIPRIORITY+Q2.0_CLERK
 +Q2.0_ORDERPRIORITY+Q2.0_ORDERDATE
 +Q2.0_TOTALPRICE+Q2.0_ORDERSTATUS+Q2.0_CUSTKEY

 9) From Operator #7

 Estimated number of rows: 3.99907
 Number of columns: 16

Subquery predicate ID: Not Applicable

Column Names:

+Q1.L_ORDERKEY(A)+Q1.L_COMMENT+Q1.L_SHIPMODE
+Q1.L_SHIPINSTRUCT+Q1.L_RECEIPTDATE
+Q1.L_COMMITDATE+Q1.L_SHIPDATE+Q1.L_LINESTATUS
+Q1.L_RETURNFLAG+Q1.L_TAX+Q1.L_DISCOUNT
+Q1.L_EXTENDEDPRICE+Q1.L_QUANTITY
+Q1.L_LINENUMBER+Q1.L_SUPPKEY+Q1.L_PARTKEY

Output Streams:

10) To Operator #1

Estimated number of rows: 5.9986e+07
Number of columns: 25
Subquery predicate ID: Not Applicable

Column Names:

+Q3.L_COMMENT+Q3.L_SHIPMODE+Q3.L_SHIPINSTRUCT
+Q3.L_RECEIPTDATE+Q3.L_COMMITDATE
+Q3.L_SHIPDATE+Q3.L_LINESTATUS+Q3.L_RETURNFLAG
+Q3.L_TAX+Q3.L_DISCOUNT+Q3.L_EXTENDEDPRICE
+Q3.L_QUANTITY+Q3.L_LINENUMBER+Q3.L_SUPPKEY
+Q3.L_PARTKEY+Q3.L_ORDERKEY+Q3.O_COMMENT
+Q3.O_SHIPPRIORITY+Q3.O_CLERK
+Q3.O_ORDERPRIORITY+Q3.O_ORDERDATE
+Q3.O_TOTALPRICE+Q3.O_ORDERSTATUS+Q3.O_CUSTKEY
+Q3.O_ORDERKEY

3) TBSCAN: (Table Scan)

Cumulative Total Cost: 8.27059e+06
Cumulative CPU Cost: 1.60961e+11
Cumulative I/O Cost: 1.32619e+06
Cumulative Re-Total Cost: 805664
Cumulative Re-CPU Cost: 5.1045e+10
Cumulative Re-I/O Cost: 441177
Cumulative First Row Cost: 7.49133e+06
Estimated Bufferpool Buffers: 441177
Remote communication cost: 9.60268e+06

Arguments:

JN INPUT: (Join input leg)
OUTER

MAXPAGES: (Maximum pages for prefetch)
ALL
PREFETCH: (Type of Prefetch)
SEQUENTIAL
SCANDIR : (Scan Direction)
FORWARD

Input Streams:

3) From Operator #4

Estimated number of rows: 1.5e+07
Number of columns: 9
Subquery predicate ID: Not Applicable

Column Names:

+Q2.0_ORDERKEY(A)+Q2.0_COMMENT
+Q2.0_SHIPRIORITY+Q2.0_CLERK
+Q2.0_ORDERPRIORITY+Q2.0_ORDERDATE
+Q2.0_TOTALPRICE+Q2.0_ORDERSTATUS+Q2.0_CUSTKEY

Output Streams:

4) To Operator #2

Estimated number of rows: 1.5e+07
Number of columns: 9
Subquery predicate ID: Not Applicable

Column Names:

+Q2.0_ORDERKEY(A)+Q2.0_COMMENT
+Q2.0_SHIPRIORITY+Q2.0_CLERK
+Q2.0_ORDERPRIORITY+Q2.0_ORDERDATE
+Q2.0_TOTALPRICE+Q2.0_ORDERSTATUS+Q2.0_CUSTKEY

4) SORT : (Sort)

Cumulative Total Cost: 7.46492e+06
Cumulative CPU Cost: 1.09916e+11
Cumulative I/O Cost: 885017
Cumulative Re-Total Cost: 0
Cumulative Re-CPU Cost: 0
Cumulative Re-I/O Cost: 441177
Cumulative First Row Cost: 7.46492e+06
Estimated Bufferpool Buffers: 885017
Remote communication cost: 9.60268e+06

Arguments:

DUPLWARN: (Duplicates Warning flag)
FALSE
NUMROWS : (Estimated number of rows)
15000000
ROWWIDTH: (Estimated width of rows)
112
SORTKEY : (Sort Key column)
1: Q2.0_ORDERKEY(A)
SPILLED : (Pages spilled to bufferpool or disk)
441177
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

2) From Operator #5

Estimated number of rows: 1.5e+07
Number of columns: 9
Subquery predicate ID: Not Applicable

Column Names:

+Q2.0_COMMENT+Q2.0_SHIPPRIORITY+Q2.0_CLERK
+Q2.0_ORDERPRIORITY+Q2.0_ORDERDATE
+Q2.0_TOTALPRICE+Q2.0_ORDERSTATUS+Q2.0_CUSTKEY
+Q2.0_ORDERKEY

Output Streams:

3) To Operator #3

Estimated number of rows: 1.5e+07
Number of columns: 9
Subquery predicate ID: Not Applicable

Column Names:

+Q2.0_ORDERKEY(A)+Q2.0_COMMENT
+Q2.0_SHIPPRIORITY+Q2.0_CLERK
+Q2.0_ORDERPRIORITY+Q2.0_ORDERDATE
+Q2.0_TOTALPRICE+Q2.0_ORDERSTATUS+Q2.0_CUSTKEY

5) SHIP : (Ship)
 Cumulative Total Cost: 1.74862e+06
 Cumulative CPU Cost: 2.24392e+10
 Cumulative I/O Cost: 443840
 Cumulative Re-Total Cost: 1.74862e+06
 Cumulative Re-CPU Cost: 2.24392e+10
 Cumulative Re-I/O Cost: 443840
 Cumulative First Row Cost: 25.0079
 Estimated Bufferpool Buffers: 443840
 Remote communication cost:9.60268e+06

Arguments:

CSERQY : (Remote common subexpression)
 FALSE

DSTSEVER: (Destination (ship to) server)
 - (NULL).

RMTQTX : (Remote statement)

SELECT AO."O_ORDERKEY", AO."O_CUSTKEY", AO."O_ORDERSTATUS",
 AO."O_TOTALPRICE", AO."O_ORDERDATE", AO."O_ORDERPRIORITY", AO."O_CLERK",
 AO."O_SHIPPRIORITY", AO."O_COMMENT" FROM "TPCD"."ORDERS" AO FOR READ ONLY

SRCSEVER: (Source (ship from) server)

DB2SERV

STREAM : (Remote stream)
 FALSE

Input Streams:

1) From Object DB2.ORDERS

Estimated number of rows: 1.5e+07

Number of columns: 10

Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$+Q2.O_COMMENT+Q2.O_SHIPPRIORITY

+Q2.O_CLERK+Q2.O_ORDERPRIORITY+Q2.O_ORDERDATE

+Q2.O_TOTALPRICE+Q2.O_ORDERSTATUS+Q2.O_CUSTKEY

+Q2.O_ORDERKEY

Output Streams:

2) To Operator #4

Estimated number of rows: 1.5e+07

Number of columns: 9

Subquery predicate ID: Not Applicable

Column Names:

+Q2.0_COMMENT+Q2.0_SHIPRIORITY+Q2.0_CLERK
+Q2.0_ORDERPRIORITY+Q2.0_ORDERDATE
+Q2.0_TOTALPRICE+Q2.0_ORDERSTATUS+Q2.0_CUSTKEY
+Q2.0_ORDERKEY

7) FILTER: (Filter)

Cumulative Total Cost: 4.41229e+07
Cumulative CPU Cost: 7.67615e+11
Cumulative I/O Cost: 6.4405e+06
Cumulative Re-Total Cost: 4.935e+06
Cumulative Re-CPU Cost: 2.33455e+11
Cumulative Re-I/O Cost: 2.72664e+06
Cumulative First Row Cost: 3.93068e+07
Estimated Bufferpool Buffers: 2.72664e+06
Remote communication cost:4.33706e+07

Arguments:

JN INPUT: (Join input leg)
INNER

Predicates:

2) Residual Predicate
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 6.66667e-08

Predicate Text:

(Q2.0_ORDERKEY = Q1.L_ORDERKEY)

Input Streams:

8) From Operator #8

Estimated number of rows: 5.99861e+07
Number of columns: 16
Subquery predicate ID: Not Applicable

Column Names:

+Q1.L_ORDERKEY(A)+Q1.L_COMMENT+Q1.L_SHIPMODE

```

+Q1.L_SHIPINSTRUCT+Q1.L_RECEIPTDATE
+Q1.L_COMMITDATE+Q1.L_SHIPDATE+Q1.L_LINESTATUS
+Q1.L_RETURNFLAG+Q1.L_TAX+Q1.L_DISCOUNT
+Q1.L_EXTENDEDPRICE+Q1.L_QUANTITY
+Q1.L_LINENUMBER+Q1.L_SUPPKEY+Q1.L_PARTKEY

```

Output Streams:

9) To Operator #2

```

Estimated number of rows: 3.99907
Number of columns: 16
Subquery predicate ID: Not Applicable

```

Column Names:

```

+Q1.L_ORDERKEY(A)+Q1.L_COMMENT+Q1.L_SHIPMODE
+Q1.L_SHIPINSTRUCT+Q1.L_RECEIPTDATE
+Q1.L_COMMITDATE+Q1.L_SHIPDATE+Q1.L_LINESTATUS
+Q1.L_RETURNFLAG+Q1.L_TAX+Q1.L_DISCOUNT
+Q1.L_EXTENDEDPRICE+Q1.L_QUANTITY
+Q1.L_LINENUMBER+Q1.L_SUPPKEY+Q1.L_PARTKEY

```

8) TBSCAN: (Table Scan)

```

Cumulative Total Cost: 4.41229e+07
Cumulative CPU Cost: 7.67615e+11
Cumulative I/O Cost: 6.4405e+06
Cumulative Re-Total Cost: 4.935e+06
Cumulative Re-CPU Cost: 2.33455e+11
Cumulative Re-I/O Cost: 2.72664e+06
Cumulative First Row Cost: 3.93068e+07
Estimated Bufferpool Buffers: 2.72664e+06
Remote communication cost:4.33706e+07

```

Arguments:

```

MAXPAGES: (Maximum pages for prefetch)
ALL
PREFETCH: (Type of Prefetch)
SEQUENTIAL
SCANDIR : (Scan Direction)
FORWARD

```

Input Streams:

7) From Operator #9

Estimated number of rows: 5.99861e+07
Number of columns: 16
Subquery predicate ID: Not Applicable

Column Names:

+Q1.L_ORDERKEY(A)+Q1.L_COMMENT+Q1.L_SHIPMODE
+Q1.L_SHIPINSTRUCT+Q1.L_RECEIPTDATE
+Q1.L_COMMITDATE+Q1.L_SHIPDATE+Q1.L_LINESTATUS
+Q1.L_RETURNFLAG+Q1.L_TAX+Q1.L_DISCOUNT
+Q1.L_EXTENDEDPRICE+Q1.L_QUANTITY
+Q1.L_LINENUMBER+Q1.L_SUPPKEY+Q1.L_PARTKEY

Output Streams:

8) To Operator #7

Estimated number of rows: 5.99861e+07
Number of columns: 16
Subquery predicate ID: Not Applicable

Column Names:

+Q1.L_ORDERKEY(A)+Q1.L_COMMENT+Q1.L_SHIPMODE
+Q1.L_SHIPINSTRUCT+Q1.L_RECEIPTDATE
+Q1.L_COMMITDATE+Q1.L_SHIPDATE+Q1.L_LINESTATUS
+Q1.L_RETURNFLAG+Q1.L_TAX+Q1.L_DISCOUNT
+Q1.L_EXTENDEDPRICE+Q1.L_QUANTITY
+Q1.L_LINENUMBER+Q1.L_SUPPKEY+Q1.L_PARTKEY

9) SORT : (Sort)

Cumulative Total Cost: 3.91879e+07
Cumulative CPU Cost: 5.34161e+11
Cumulative I/O Cost: 3.71386e+06
Cumulative Re-Total Cost: 0
Cumulative Re-CPU Cost: 0
Cumulative Re-I/O Cost: 2.72664e+06
Cumulative First Row Cost: 3.91879e+07
Estimated Bufferpool Buffers: 3.71386e+06
Remote communication cost:4.33706e+07

Arguments:

DUPLWARN: (Duplicates Warning flag)

FALSE

NUMROWS : (Estimated number of rows)

59986052

ROWWIDTH: (Estimated width of rows)
176
SORTKEY : (Sort Key column)
1: Q1.L_ORDERKEY(A)
SPILLED : (Pages spilled to bufferpool or disk)
2.72664e+06
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

6) From Operator #10

Estimated number of rows: 5.99861e+07
Number of columns: 16
Subquery predicate ID: Not Applicable

Column Names:

+Q1.L_COMMENT+Q1.L_SHIPMODE+Q1.L_SHIPINSTRUCT
+Q1.L_RECEIPTDATE+Q1.L_COMMITDATE
+Q1.L_SHIPDATE+Q1.L_LINESTATUS+Q1.L_RETURNFLAG
+Q1.L_TAX+Q1.L_DISCOUNT+Q1.L_EXTENDEDPRICE
+Q1.L_QUANTITY+Q1.L_LINENUMBER+Q1.L_SUPPKEY
+Q1.L_PARTKEY+Q1.L_ORDERKEY

Output Streams:

7) To Operator #8

Estimated number of rows: 5.99861e+07
Number of columns: 16
Subquery predicate ID: Not Applicable

Column Names:

+Q1.L_ORDERKEY(A)+Q1.L_COMMENT+Q1.L_SHIPMODE
+Q1.L_SHIPINSTRUCT+Q1.L_RECEIPTDATE
+Q1.L_COMMITDATE+Q1.L_SHIPDATE+Q1.L_LINESTATUS
+Q1.L_RETURNFLAG+Q1.L_TAX+Q1.L_DISCOUNT
+Q1.L_EXTENDEDPRICE+Q1.L_QUANTITY
+Q1.L_LINENUMBER+Q1.L_SUPPKEY+Q1.L_PARTKEY

10) SHIP : (Ship)
Cumulative Total Cost: 3.92801e+06

Cumulative CPU Cost: 1.21469e+11
Cumulative I/O Cost: 987218
Cumulative Re-Total Cost: 3.92801e+06
Cumulative Re-CPU Cost: 1.21469e+11
Cumulative Re-I/O Cost: 987218
Cumulative First Row Cost: 25.0079
Estimated Bufferpool Buffers: 987218
Remote communication cost:4.33706e+07

Arguments:

CSERQY : (Remote common subexpression)
FALSE

DSTSEVER: (Destination (ship to) server)
- (NULL).

RMTQTX : (Remote statement)

SELECT AO."L_ORDERKEY", AO."L_PARTKEY", AO."L_SUPPKEY",
AO."L_LINENUMBER", AO."L_QUANTITY", AO."L_EXTENDEDPRI", AO."L_DISCOUNT",
AO."L_TAX", AO."L_RETURNFLAG", AO."L_LINESTATUS", AO."L_SHIPDATE",
AO."L_COMMITDATE", AO."L_RECEIPTDATE", AO."L_SHIPINSTRUCT", AO."L_SHIPMODE",
AO."L_COMMENT" FROM "IITEST"."LINEITEM" AO

SRCSEVER: (Source (ship from) server)
ORASERV

STREAM : (Remote stream)
FALSE

Input Streams:

5) From Object ORA.LINEITEM

Estimated number of rows: 5.99861e+07
Number of columns: 17
Subquery predicate ID: Not Applicable

Column Names:

+Q1.\$RID\$+Q1.L_COMMENT+Q1.L_SHIPMODE
+Q1.L_SHIPINSTRUCT+Q1.L_RECEIPTDATE
+Q1.L_COMMITDATE+Q1.L_SHIPDATE+Q1.L_LINESTATUS
+Q1.L_RETURNFLAG+Q1.L_TAX+Q1.L_DISCOUNT
+Q1.L_EXTENDEDPRI+Q1.L_QUANTITY
+Q1.L_LINENUMBER+Q1.L_SUPPKEY+Q1.L_PARTKEY
+Q1.L_ORDERKEY

Output Streams:

6) To Operator #9

Estimated number of rows: 5.99861e+07
Number of columns: 16
Subquery predicate ID: Not Applicable

Column Names:

+Q1.L_COMMENT+Q1.L_SHIPMODE+Q1.L_SHIPINSTRUCT
+Q1.L_RECEIPTDATE+Q1.L_COMMITDATE
+Q1.L_SHIPDATE+Q1.L_LINESTATUS+Q1.L_RETURNFLAG
+Q1.L_TAX+Q1.L_DISCOUNT+Q1.L_EXTENDEDPRICE
+Q1.L_QUANTITY+Q1.L_LINENUMBER+Q1.L_SUPPKEY
+Q1.L_PARTKEY+Q1.L_ORDERKEY

Objects Used in Access Plan:

Schema: DB2

Name: ORDERS

Type: Nickname

Time of creation: 2004-08-04-21.28.22.800997

Last statistics update:

Number of columns: 9

Number of rows: 15000000

Width of rows: 113

Number of buffer pool pages: 443840

Distinct row values: No

Tablespace name:

Tablespace overhead: 24.100000

Tablespace transfer rate: 0.900000

Source for statistics: Single Node

Prefetch page count: 32

Container extent page count: 32

Schema: ORA

Name: LINEITEM

Type: Nickname

Time of creation: 2004-08-04-22.01.51.518653

Last statistics update: 2004-08-04-22.02.08.829324

Number of columns: 16

Number of rows: 59986052

Width of rows: 177

Number of buffer pool pages: 987218

Distinct row values: No

Tablespace name:

Tablespace overhead: 24.100000

Tablespace transfer rate: 0.900000

Source for statistics: Single Node

Prefetch page count: 32

The RMTQTXT fields of SHIP operators 5 and 10 in Example 4-40 on page 242 contain the following remote SQL fragment text:

– SHIP operator 5 fragment

```
SELECT AO."O_ORDERKEY", AO."O_CUSTKEY", AO."O_ORDERSTATUS",
AO."O_TOTALPRICE", AO."O_ORDERDATE", AO."O_ORDERPRIORITY", AO."O_CLERK",
AO."O_SHIPPRIORITY", AO."O_COMMENT" FROM "TPCD"."ORDERS" AO FOR READ
ONLY
```

– SHIP operator 10 fragment

```
SELECT AO."L_ORDERKEY", AO."L_PARTKEY", AO."L_SUPPKEY",
AO."L_LINENUMBER", AO."L_QUANTITY", AO."L_EXTENDEDPRICE",
AO."L_DISCOUNT", AO."L_TAX", AO."L_RETURNFLAG", AO."L_LINESTATUS",
AO."L_SHIPDATE", AO."L_COMMITDATE", AO."L_RECEIPTDATE",
AO."L_SHIPINSTRUCT", AO."L_SHIPMODE", AO."L_COMMENT" FROM
"IITEST"."LINEITEM" AO
```

These remote SQL fragments are located in the dynamic cache output shown in Example 4-39 on page 240, and the following information can be gathered:

- **Number of executions** is 1 for the user-entered query as well as the two remote SQL fragments.
- **Total execution time (sec.ms)**, which is 3338.542900 seconds for the user-entered query, and 1870.848680 seconds and 333.479573 seconds for the two remote fragments, respectively.
- **Total user cpu time (sec.ms)** and **Total system cpu time (sec.ms)**, which is 1070.300000 and 58.810000 for the user-entered query, and zero for the remote SQL fragments.
- **Rows read** is 59986052 for the user-entered query, which is the number of rows returned to the user, and 59986052 rows and 15000000 rows for the two remote SQL fragments, respectively, which indicate the number of rows returned to the federated server from the remote data sources.
- Other fields of interest include **Statement sorts**, **Statement sort overflows**, and **Total sort time**, which only apply to the user-entered query, and all have values 2, 2, and 479027 milliseconds, respectively.
- The buffer pool hit ratio appears to be very good.

Note: To obtain the average elapsed and CPU times, as well as the number of rows returned, you must divide the values shown by the **Number of executions**.

We can derive the following information from these metrics:

- The average number of rows returned from the remote data source to the federated server are $(59986052 / 1) = 59986052$ and $(15000000 / 1) = 15000000$, respectively.
- The average elapsed time for the user query is $(3338.542900 / 1) = 3338.542900$ seconds, while that of the remote SQL fragments are $(1870.848680 / 1) = 1870.848680$ seconds and $(333.479573 / 1) = 333.479573$, respectively.

Attention: In our example, given that the total query elapsed time is 3338.542900 seconds, it is clear that two-thirds of the elapsed time of the query is spent at the remote data source, and one-third at the federated server. The percent of time spent at the remote data sources is $((1870.848680 + 333.479573) / 3338.542900) = 66\%$. The time spent at the federated server is $(3338.542900 - (1870.848680 + 333.479573)) = 1134.214647$ seconds.

The sort time of 479.027 seconds is high as compared to the total time (1134.214647 seconds) spent at the federated server, as is the occurrence of sort overflows.

This performance problem therefore needs to be investigated at both the remote data source and the federated server, as described in 4.2.4, “Federated server related” on page 152; and 4.2.5, “Remote data source related” on page 161.

► Remote data source related.

The remote SQL fragments for the problem query show every row in the orders and line item tables being returned to the federated server. Such a scan of millions of rows is time consuming, and appropriate tuning strategies have to be adopted at these remote data sources to improve scan performance.

► Federated server related.

As discussed in 4.2.4, “Federated server related” on page 152, poor performance at the federated server may be caused by a number of factors such as statistics, index information, pushdown, joins, and parallelism.

Each of these factors need to be evaluated in turn to determine the root cause of a performance problem.

- A review of the number of rows estimated by the DB2 optimizer in Example 4-40 on page 242 as being returned from the remote data sources to the federated server, and the actual number of rows returned (**Rows read**) from each remote data source in Example 4-39 on page 240 shows no discrepancy, and hence no investigation is needed.

- A review of the remote SQL fragment texts and the original statement (Example 4-40 on page 242), which is a join of nicknames across two different remote data sources, shows that there are no predicates that need to be investigated from a pushdown perspective.
- The join predicate has to be executed at the federated server because the joined columns belong to nicknames referencing different remote data sources.
- **db2exfmt** output in Example 4-40 on page 242 shows the merge scan join (MSJOIN operator 2) being performed with an estimated 1.5e+07 rows from DB2.ORDERS outer table with an estimated 3.99907 rows per row in the inner table ORA.LINEITEM containing 5.99861e+07 rows. Since the merge scan requires the two tables to be in sorted order, sorts are indicated by SORT operator 4 on the DB2.ORDERS table and SORT operator 9 on the ORA.LINEITEM table.

As indicated earlier, the **Total sort time** of 479.027 seconds and the **Sort overflows** of 2 in Example 4-39 on page 240 merit investigation. However, the sum of (**Total user cpu time (sec.ms)** + **Total system CPU time (sec.ms)**) is = (1070.300000 + 58.810000) = 1129.11 seconds. This is almost equal to the 1134.214647 seconds spent at the federated server and is probably entirely due to the processing of 5.99861e+07 rows. Unless the number of rows is reduced, the processing time at the federated server cannot be reduced.

Given the equality predicates (D.ORDERKEY = L.L_ORDERKEY) and the characteristics of the tables involved in the join (one very large table of 5.99861e+07 rows being joined with a smaller table of approximately 1.5e+07 rows), we investigated why a hash join was not chosen (see Table 4-2 on page 158) as the preferred access path since sort costs would potentially be lower (depending upon other considerations such as SORTHEAP and the buffer pool).

To determine whether there was a data type mismatch inhibiting the consideration of a hash join as an optimal access path, we ran the query against the SYSCAT.COLUMNS catalog view, as shown in Example 4-41, to determine the data types of the join columns.

Example 4-41 Data types for nickname join columns before ALTER

```
connect to fedserv
```

Database Connection Information

```
Database server          = DB2/AIX64 8.2.0
SQL authorization ID     = DB2I64
Local database alias     = FEDSERV
```

```
select substr(tabschema,1,8) as tabschema, substr(tabname,1,8) as tabname,
substr(colname,1,15) as colname, substr(typename,1,8) as typename, length,
scale from syscat.columns where (tabschema = 'ORA' and tabname = 'LINEITEM' or
tabschema = 'DB2' and tabname = 'ORDERS') and colname like '%ORDERKEY'
```

TABSCHEMA	TABNAME	COLNAME	TYPENAME	LENGTH	SCALE
ORA	LINEITEM	L_ORDERKEY	DECIMAL	10	0
DB2	ORDERS	O_ORDERKEY	INTEGER	4	0

2 record(s) selected

Example 4-41 shows the join columns to have different data types—L_ORDERKEY on the LINEITEM nickname is DECIMAL(10,0), while O_ORDERKEY on the ORDERS nickname is INTEGER.

Note: This is a common issue with Oracle nicknames because Oracle DBAs tend to define all Oracle numeric columns as NUMBER(), regardless of whether they are used to store integers, decimals, or real values. In order to ensure that the nickname data type in DB2 II is able to hold all possible values that can be contained within an Oracle NUMBER column, the default type mapping employed is to a DECIMAL(10,0) column. Now, if it is known that the remote column will only ever hold integers, then it is safe to change this mapping from DECIMAL to INTEGER. This problem is more pronounced with Oracle data sources because it is very common for the tables to have columns defined as NUMBER, rather than using the more precise types like INTEGER, DECIMAL, etc.

Root cause of the problem

It appears that there are at least two possible causes of the performance problem:

- Mismatch of data types possibly resulting in a less optimal join strategy (merge scan being chosen instead of a hash join)—possibly resulting in sort costs
- Scan performance at the remote data sources

Apply best practices

Since tuning the remote data sources is beyond the scope of this publication, we focus here on the mismatch of data types problem.

We recommend the following steps to address the data type mismatch problem:

1. Capture HIGH2KEY and LOW2KEY statistics for the join column whose data types are to be altered.

This needs to be done *before* altering the data type because these values are lost after the ALTER NICKNAME statement is executed.

Example 4-42 shows how to capture the cardinality and HIGH2KEY and LOW2KEY statistics for the joins column.

Example 4-42 Statistics for nickname join columns before ALTER

connect to fedserv

Database Connection Information

Database server = DB2/AIX64 8.2.0
SQL authorization ID = DB2I64
Local database alias = FEDSERV

select substr(tabschema,1,4) as tabschema, substr(tabname,1,8) as tabname, card
from syscat.tables where (tabschema = 'ORA' and tabname = 'LINEITEM' or
tabschema = 'DB2' and tabname = 'ORDERS')

TABSCHEMA	TABNAME	CARD
DB2	ORDERS	15000000
ORA	LINEITEM	59986052

2 record(s) selected.

select substr(tabschema,1,4) as tabschema, substr(tabname,1,8) as tabname,
substr(colname,1,10) as colname, colcard, substr(high2key,1,20) as high2key,
substr(low2key,1,20) as low2key from sysstat.columns where (tabschema = 'ORA'
and tabname = 'LINEITEM' or tabschema = 'DB2' and tabname = 'ORDERS') and
colname like '%ORDERKEY'

TABSCHEMA	TABNAME	COLNAME	COLCARD	HIGH2KEY	LOW2KEY
DB2	ORDERS	O_ORDERKEY	15000000	59999975	2
ORA	LINEITEM	L_ORDERKEY	15000000	59999975	2

2 record(s) selected.

2. Alter the data type of one of the join columns using the ALTER NICKNAME statement.

Example 4-43 shows the L_ORDERKEY column data type being changed to an INTEGER data type to match that of the O_ORDERKEY.

Example 4-43 ALTER NICKNAME statement

connect to fedserv

Database Connection Information

Database server = DB2/AIX64 8.2.0
SQL authorization ID = DB2I64
Local database alias = FEDSERV

alter nickname ora.lineitem alter column l_orderkey local type integer
DB20000I The SQL command completed successfully.

Example 4-44 on page 262 verifies that the data types of the two columns are indeed identical now and have the INTEGER data type.

Example 4-44 Data types for nickname join columns after ALTER

connect to fedserv

Database Connection Information

Database server = DB2/AIX64 8.2.0
SQL authorization ID = DB2I64
Local database alias = FEDSERV

select substr(tabschema,1,8) as tabschema, substr(tabname,1,8) as tabname,
substr(colname,1,15) as colname, substr(typename,1,8) as typename, length,
scale from syscat.columns where (tabschema = 'ORA' and tabname = 'LINEITEM' or
tabschema = 'DB2' and tabname = 'ORDERS') and colname like '%ORDERKEY'

TABSCHEMA	TABNAME	COLNAME	TYPENAME	LENGTH	SCALE
ORA	LINEITEM	L_ORDERKEY	INTEGER	4	0
DB2	ORDERS	O_ORDERKEY	INTEGER	4	0

2 record(s) selected.

Example 4-45 shows the loss of the HIGH2KEY and LOW2KEY values for the L_ORDERKEY column as a result of the ALTER NICKNAME statement execution.

Example 4-45 Statistics for nickname join columns after ALTER

connect to fedserv

Database Connection Information

Database server = DB2/AIX64 8.2.0

SQL authorization ID = DB2I64
Local database alias = FEDSERV

```
select substr(tabschema,1,4) as tabschema, substr(tabname,1,8) as tabname, card
from syscat.tables where (tabschema = 'ORA' and tabname = 'LINEITEM' or
tabschema = 'DB2' and tabname = 'ORDERS')
```

TABSCHEMA	TABNAME	CARD
DB2	ORDERS	15000000
ORA	LINEITEM	59986052

2 record(s) selected.

```
select substr(tabschema,1,4) as tabschema, substr(tabname,1,8) as tabname,
substr(colname,1,10) as colname, colcard, substr(high2key,1,20) as high2key,
substr(low2key,1,20) as low2key from sysstat.columns where (tabschema = 'ORA'
and tabname = 'LINEITEM' or tabschema = 'DB2' and tabname = 'ORDERS') and
colname like '%ORDERKEY'
```

TABSCHEMA	TABNAME	COLNAME	COLCARD	HIGH2KEY	LOW2KEY
DB2	ORDERS	O_ORDERKEY	15000000	59999975	2
ORA	LINEITEM	L_ORDERKEY	15000000		

2 record(s) selected.

Note: When altering a nickname's local data types, it is the responsibility of the DBA to ensure that the local data type is able to hold all the values that are stored in the remote column. Failure to do so could result in inconsistent results being returned to the application.

3. Re-establish the correct values for HIGH2KEY and LOW2KEY for the L_ORDERKEY column.

Example 4-46 shows the SQL statement for updating the HIGH2KEY and LOW2KEY values for the L_ORDERKEY with the values that we captured in Example 4-42 on page 261 prior to altering the data type.

Example 4-46 Manually update HIGH2KEY and LOW2KEY values

connect to fedserv

Database Connection Information

Database server = DB2/AIX64 8.2.0

SQL authorization ID = DB2I64
Local database alias = FEDSERV

update sysstat.columns set high2key = '59999975', low2key = '2' where colname = 'L_ORDERKEY' and tabname = 'LINEITEM' and tabschema like 'ORA%'
DB20000I The SQL command completed successfully.

4. Run **db2exfmt** on the query.

Example 4-47 on page 264 shows the **db2exfmt** output after the data type mismatch was removed. The Access Plan section shows a hash join (HSJOIN operator 3) being chosen, and the Total Cost being estimated to be 5.86806e+06 timerons, which is less than the merge scan join selection estimate of 5.24115e+07 timerons in Example 4-40 on page 242.

Attention: In the final analysis, what really matters is not what the DB2 optimizer estimates to be the optimal access path based on timerons, but the actual run times experienced by the user.

The technique of altering the local type, length/precision, and scale of the nickname column is only valid if:

- ▶ The new type is compatible with the existing type.
- ▶ The new type, with its length/precision, and scale, will not cause value truncation or padding that makes results of the query invalid.

In this example, altering the nickname column local type is valid since:

- ▶ Both decimal(10,0) and integer hold numeric values that are compatible.
- ▶ Decimal(10,0) has scale=0 and so can hold only whole values, just like integer.
- ▶ Decimal(10,0) with precision=10 holds values whose maximum length (10 digits) is same as integer, so no part of the values will be truncated and there will be no padding when the decimal values are converted to integer.

Other techniques are described in “Nickname-related best practices” on page 109.

Since tuning tends to be a trial-and-error iterative process, it is more than likely that some of the options suggested could lead to other performance problems that would need to be investigated and resolved.

Example 4-47 db2exfmt of problem query after fixing mismatched data types

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002

***** EXPLAIN INSTANCE *****

DB2_VERSION: 08.02.0
SOURCE_NAME: SQLC2E05
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-08-04-21.42.08.693004
EXPLAIN_REQUESTER: DB2I64

Database Context:

Parallelism: None
CPU Speed: 5.392596e-07
Comm Speed: 100
Buffer Pool size: 75000
Sort Heap size: 100000
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 640

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 203 -----

QUERYNO: 1
QUERYTAG:
Statement Type: Select
Updatable: No
Deletable: No
Query Degree: 1

Original Statement:

**select count(*)
from db2.orders d, ora.lineitem l
where d.o_orderkey = l.l_orderkey**

Optimized Statement:

```
-----  
SELECT Q4.$C0  
FROM  
  (SELECT COUNT(* )  
   FROM  
     (SELECT $RID$  
      FROM ORA.LINEITEM AS Q1, DB2.ORDERS AS Q2  
      WHERE (Q2.O_ORDERKEY = Q1.L_ORDERKEY)) AS Q3) AS Q4
```

Access Plan:

```
-----  
Total Cost: 5.86806e+06  
Query Degree:1  
  
      Rows  
      RETURN  
      ( 1)  
      Cost  
      I/O  
      |  
      1  
      GRPBY  
      ( 2)  
      5.86806e+06  
      1.45897e+06  
      |  
      5.99861e+07  
      HSJOIN  
      ( 3)  
      5.85997e+06  
      1.45897e+06  
      /-----+-----\  
      5.99861e+07      1.5e+07  
      SHIP            SHIP  
      ( 4)            ( 6)  
      3.8814e+06      1.74195e+06  
      987218          443840  
      |              |  
      5.99861e+07      1.5e+07  
      NICKNM: ORA      NICKNM: DB2  
      LINEITEM          ORDERS
```

1) RETURN: (Return Result)

Cumulative Total Cost: 5.86806e+06
Cumulative CPU Cost: 7.96331e+10
Cumulative I/O Cost: 1.45897e+06
Cumulative Re-Total Cost: 5.86806e+06
Cumulative Re-CPU Cost: 7.96331e+10
Cumulative Re-I/O Cost: 1.45897e+06
Cumulative First Row Cost: 5.86806e+06
Estimated Bufferpool Buffers: 1.44571e+06
Remote communication cost:3.88231e+07

Arguments:

BLDLEVEL: (Build level)
DB2 v8.1.1.64 : s040509
ENVVAR : (Environment Variable)
DB2_EXTENDED_OPTIMIZATION = ON
STMHEAP: (Statement heap size)
8192

Input Streams:

6) From Operator #2

Estimated number of rows: 1
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q5.\$C0

2) GRPBY : (Group By)
Cumulative Total Cost: 5.86806e+06
Cumulative CPU Cost: 7.96331e+10
Cumulative I/O Cost: 1.45897e+06
Cumulative Re-Total Cost: 5.86806e+06
Cumulative Re-CPU Cost: 7.96331e+10
Cumulative Re-I/O Cost: 1.45897e+06
Cumulative First Row Cost: 5.86806e+06
Estimated Bufferpool Buffers: 1.44571e+06
Remote communication cost:3.88231e+07

Arguments:

AGGMODE : (Aggregation Mode)
COMPLETE
GROUPBYC: (Group By columns)
FALSE

GROUPBYN: (Number of Group By columns)
0
ONEFETCH: (One Fetch flag)
FALSE

Input Streams:

5) From Operator #3

Estimated number of rows: 5.99861e+07
Number of columns: 0
Subquery predicate ID: Not Applicable

Output Streams:

6) To Operator #1

Estimated number of rows: 1
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q5.\$C0

3) HSJOIN: (Hash Join)

Cumulative Total Cost: 5.85997e+06
Cumulative CPU Cost: 6.46366e+10
Cumulative I/O Cost: 1.45897e+06
Cumulative Re-Total Cost: 5.85997e+06
Cumulative Re-CPU Cost: 6.46366e+10
Cumulative Re-I/O Cost: 1.45897e+06
Cumulative First Row Cost: 5.85997e+06
Estimated Bufferpool Buffers: 1.44571e+06
Remote communication cost:3.88231e+07

Arguments:

BITFLTR : (Hash Join Bit Filter used)
FALSE
EARLYOUT: (Early Out flag)
LEFT
HASHCODE: (Hash Code Size)
24 BIT
TEMPSIZE: (Temporary Table Page Size)
4096

Predicates:

- 2) Predicate used in Join
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 6.66667e-08

Predicate Text:

(Q2.0_ORDERKEY = Q1.L_ORDERKEY)

Input Streams:

- 2) From Operator #4

Estimated number of rows: 5.99861e+07
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q1.L_ORDERKEY

- 4) From Operator #6

Estimated number of rows: 1.5e+07
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.0_ORDERKEY

Output Streams:

- 5) To Operator #2

Estimated number of rows: 5.99861e+07
Number of columns: 0
Subquery predicate ID: Not Applicable

- 4) SHIP : (Ship)

Cumulative Total Cost: 3.8814e+06
Cumulative CPU Cost: 3.50356e+10
Cumulative I/O Cost: 987218
Cumulative Re-Total Cost: 3.8814e+06

Cumulative Re-CPU Cost: 3.50356e+10
Cumulative Re-I/O Cost: 987218
Cumulative First Row Cost: 25.0079
Estimated Bufferpool Buffers: 987218
Remote communication cost:3.10711e+07

Arguments:

CSERQY : (Remote common subexpression)
FALSE
DSTSEVER: (Destination (ship to) server)
- (NULL).
JN INPUT: (Join input leg)
OUTER
RMTQTX : (Remote statement)
SELECT AO."L_ORDERKEY" FROM "IITEST"."LINEITEM" AO
SRCSEVER: (Source (ship from) server)
ORASERV
STREAM : (Remote stream)
FALSE

Input Streams:

1) From Object ORA.LINEITEM

Estimated number of rows: 5.99861e+07
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q1.\$RID\$+Q1.L_ORDERKEY

Output Streams:

2) To Operator #3

Estimated number of rows: 5.99861e+07
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q1.L_ORDERKEY

6) SHIP : (Ship)
Cumulative Total Cost: 1.74195e+06

Cumulative CPU Cost: 1.00656e+10
Cumulative I/O Cost: 443840
Cumulative Re-Total Cost: 1.74195e+06
Cumulative Re-CPU Cost: 1.00656e+10
Cumulative Re-I/O Cost: 443840
Cumulative First Row Cost: 25.0079
Estimated Bufferpool Buffers: 443840
Remote communication cost:7.75204e+06

Arguments:

CSERQY : (Remote common subexpression)
FALSE
DSTSEVER: (Destination (ship to) server)
- (NULL).
JN INPUT: (Join input leg)
INNER
RMTQTX : (Remote statement)
SELECT A0."O_ORDERKEY" FROM "TPCD"."ORDERS" A0 FOR READ ONLY
SRCSEVER: (Source (ship from) server)
DB2SERV
STREAM : (Remote stream)
FALSE

Input Streams:

3) From Object DB2.ORDERS

Estimated number of rows: 1.5e+07
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$+Q2.O_ORDERKEY

Output Streams:

4) To Operator #3

Estimated number of rows: 1.5e+07
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.O_ORDERKEY

Objects Used in Access Plan:

Schema: DB2

Name: ORDERS

Type: Nickname

Time of creation: 2004-08-04-21.28.22.800997

Last statistics update:

Number of columns: 9

Number of rows: 15000000

Width of rows: 16

Number of buffer pool pages: 443840

Distinct row values: No

Tablespace name:

Tablespace overhead: 24.100000

Tablespace transfer rate: 0.900000

Source for statistics: Single Node

Prefetch page count: 32

Container extent page count: 32

Schema: ORA

Name: LINEITEM

Type: Nickname

Time of creation: 2004-08-04-21.35.47.856522

Last statistics update: 2004-08-04-21.41.05.553007

Number of columns: 16

Number of rows: 59986052

Width of rows: 35

Number of buffer pool pages: 987218

Distinct row values: No

Tablespace name:

Tablespace overhead: 24.100000

Tablespace transfer rate: 0.900000

Source for statistics: Single Node

Prefetch page count: 32

Container extent page count: 32

4.4.6 Pushdown problems

The DB2_MAXIMAL_PUSHDOWN server option allows the user or DBA to direct the optimizer to choose an access plan based on cost or have as much query processing as possible be performed by the remote data sources. With the default setting of DB2_MAXIMAL_PUSHDOWN of 'N', the optimizer is directed to choose an access plan based on cost optimization.

In this scenario, we diagnose how the default setting of the DB2_MAXIMAL_PUSHDOWN server option may inhibit a particular federated query (about which the user has additional domain expertise) from achieving superior performance.

Triggering event

Users complained about poor response times with a specific query.

Hypotheses and validations

Here again, since these were user complaints about the performance of a specific query, as per Figure 4-3 on page 122, we decided to enter the DB2 hypotheses hierarchy shown in Figure 4-1 on page 117 and described in Example 4.2 on page 119, at a lower level, bypassing network- and system-related problems, and focused directly on federated application/query performance, as follows.

Hypothesis 1: Federated application/query performance

Before one can investigate the cause of a query's performance problem, one needs to identify the query in question. After identifying the query in question, one can begin diagnosing whether the performance problem is at the federated server, the remote data source, or equally divided between the two, as discussed in Example 4.2 on page 119.

- Identify the application and query.

In this case the user specifically identified the query in question as being the one shown in Example 4-48. It finds which supplier should be selected to place an order for a given (brass, size 15) part in a given region (Europe) based on the minimum supplier cost. The parts, supplier, and parts supplier tables reside on DB2, while the nation and region tables reside on Oracle.

Example 4-48 Problem query

```
SELECT S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY, P_MFGR, S_ADDRESS, S_PHONE,
S_COMMENT
FROM db2.PART, db2.SUPPLIER, db2.PARTSUPP, ora.NATION, ora.REGION
WHERE P_PARTKEY = PS_PARTKEY AND
      S_SUPPKEY = PS_SUPPKEY AND
      P_SIZE = 15 AND
      P_TYPE LIKE '%BRASS' AND
      S_NATIONKEY = N_NATIONKEY AND
      R_NAME = 'EUROPE' AND
      PS_SUPPLYCOST =
        (SELECT MIN(PS_SUPPLYCOST)
         FROM db2.PARTSUPP, db2.SUPPLIER, ora.NATION, ora.REGION
         WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY AND
              S_NATIONKEY = N_NATIONKEY AND
```

```

        N_REGIONKEY = R_REGIONKEY AND
        R_NAME = 'EUROPE' )
ORDER BY S_ACCTBAL DESC, N_NAME, S_NAME, P_PARTKEY
FETCH FIRST 100 ROWS ONLY

```

- Determine if the problem is at the Federated server or the data source.

Once the problem query has been identified, we need to determine if the query's performance problem is at the federated server, the remote data source, or distributed between both.

This information can be determined from a dynamic SQL snapshot, as discussed in 4.2.3, "Federated server or remote data source" on page 136. Example 4-49 is a dynamic SQL snapshot, and includes the problem query (highlighted) in the Statement text field.

Example 4-49 Dynamic SQL snapshot

get snapshot for dynamic sql on fedserv

Dynamic SQL Snapshot Result

```

Database name                = FEDSERV
Database path                 = /data1/npart/db2i64/NODE0000/SQL00001/

Number of executions         = 1
Number of compilations       = 1
Worst preparation time (ms)  = 142
Best preparation time (ms)   = 142
Internal rows deleted        = 0
Internal rows inserted       = 0
Rows read                    = 100
Internal rows updated        = 0
Rows written                 = 4668
Statement sorts              = 62833
Statement sort overflows     = 1
Total sort time              = 13
Buffer pool data logical reads = 6
Buffer pool data physical reads = 1
Buffer pool temporary data logical reads = 292
Buffer pool temporary data physical reads = 0
Buffer pool index logical reads = 7
Buffer pool index physical reads = 3
Buffer pool temporary index logical reads = 0
Buffer pool temporary index physical reads = 0
Total execution time (sec.ms)      = 80.444751
Total user cpu time (sec.ms)       = 9.920000
Total system cpu time (sec.ms)     = 1.980000

```



```

Statement text                                = SELECT S_ACCTBAL, S_NAME, N_NAME,
P_PARTKEY, P_MFGR, S_ADDRESS, S_PHONE, S_COMMENT FROM db2.PART, db2.SUPPLIER,
db2.PARTSUPP, ora.NATION, ora.REGION WHERE P_PARTKEY = PS_PARTKEY AND
S_SUPPKEY = PS_SUPPKEY AND P_SIZE = 15 AND P_TYPE LIKE '%BRASS' AND
S_NATIONKEY = N_NATIONKEY AND R_NAME = 'EUROPE' AND PS_SUPPLYCOST = (SELECT
MIN(PS_SUPPLYCOST) FROM db2.PARTSUPP, db2.SUPPLIER, ora.NATION, ora.REGION
WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY AND S_NATIONKEY =
N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY AND R_NAME = 'EUROPE' ) ORDER BY
S_ACCTBAL DESC, N_NAME, S_NAME, P_PARTKEY  FETCH FIRST 100 ROWS ONLY

```

```

Number of executions                        = 4668
Number of compilations                     = 1
Worst preparation time (ms)                = 0
Best preparation time (ms)                = 0
Internal rows deleted                     = 0
Internal rows inserted                    = 0
Rows read                                 = 4668
Internal rows updated                     = 0
Rows written                             = 0
Statement sorts                           = 0
Buffer pool data logical reads            = 0
Buffer pool data physical reads           = 0
Buffer pool temporary data logical reads = 0
Buffer pool temporary data physical reads = 0
Buffer pool index logical reads           = 0
Buffer pool index physical reads          = 0
Buffer pool temporary index logical reads = 0
Buffer pool temporary index physical reads = 0
Total execution time (sec.ms)          = 0.038954
Total user cpu time (sec.ms)              = 0.000000
Total system cpu time (sec.ms)            = 0.000000
Statement text                            = SELECT A0."N_NAME" FROM "IITEST"."NATION"
AO WHERE (:H0 = A0."N_NATIONKEY")

```

```

Number of executions                        = 100
Number of compilations                     = 1
Worst preparation time (ms)                = 0
Best preparation time (ms)                = 0
Internal rows deleted                     = 0
Internal rows inserted                    = 0
Rows read                               = 100
Internal rows updated                     = 0
Rows written                             = 0
Statement sorts                           = 0
Buffer pool data logical reads            = 0
Buffer pool data physical reads           = 0
Buffer pool temporary data logical reads = 0

```

```

Buffer pool temporary data physical reads = 0
Buffer pool index logical reads = 0
Buffer pool index physical reads = 0
Buffer pool temporary index logical reads = 0
Buffer pool temporary index physical reads = 0
Total execution time (sec.ms) = 0.001376
Total user cpu time (sec.ms) = 0.000000
Total system cpu time (sec.ms) = 0.000000
Statement text = SELECT '1' FROM "IITEST"."REGION" AO
WHERE (AO."R_NAME" = 'EUROPE
')

```

```

Number of executions = 31416
Number of compilations = 1
Worst preparation time (ms) = 0
Best preparation time (ms) = 0
Internal rows deleted = 0
Internal rows inserted = 0
Rows read = 157080
Internal rows updated = 0
Rows written = 0
Statement sorts = 0
Buffer pool data logical reads = 0
Buffer pool data physical reads = 0
Buffer pool temporary data logical reads = 0
Buffer pool temporary data physical reads = 0
Buffer pool index logical reads = 0
Buffer pool index physical reads = 0
Buffer pool temporary index logical reads = 0
Buffer pool temporary index physical reads = 0
Total execution time (sec.ms) = 0.366647
Total user cpu time (sec.ms) = 0.000000
Total system cpu time (sec.ms) = 0.000000
Statement text = SELECT A1."N_NATIONKEY" FROM
"IITEST"."REGION" AO, "IITEST"."NATION" A1 WHERE (AO."R_NAME" = 'EUROPE
') AND (A1."N_REGIONKEY" = AO."R_REGIONKEY")

```

```

Number of executions = 1
Number of compilations = 1
Worst preparation time (ms) = 0
Best preparation time (ms) = 0
Internal rows deleted = 0
Internal rows inserted = 0
Rows read = 31416
Internal rows updated = 0
Rows written = 0
Statement sorts = 0
Buffer pool data logical reads = 0

```

Buffer pool data physical reads = 0
 Buffer pool temporary data logical reads = 0
 Buffer pool temporary data physical reads = 0
 Buffer pool index logical reads = 0
 Buffer pool index physical reads = 0
 Buffer pool temporary index logical reads = 0
 Buffer pool temporary index physical reads = 0
Total execution time (sec.ms) = 11.783489
 Total user cpu time (sec.ms) = 0.000000
 Total system cpu time (sec.ms) = 0.000000
Statement text = SELECT A0."P_PARTKEY", A0."P_MFGR",
 A1."PS_SUPPKEY", A1."PS_SUPPLYCOST" FROM "TPCD"."PART" A0, "TPCD"."PARTSUPP" A1
 WHERE (A0."P_SIZE" = 15) AND (A0."P_TYPE" LIKE '%BRASS') AND (A0."P_PARTKEY" =
 A1."PS_PARTKEY") FOR READ ONLY

Number of executions = 31416
 Number of compilations = 1
 Worst preparation time (ms) = 0
 Best preparation time (ms) = 0
 Internal rows deleted = 0
 Internal rows inserted = 0
Rows read = 125664
 Internal rows updated = 0
 Rows written = 0
 Statement sorts = 0
 Buffer pool data logical reads = 0
 Buffer pool data physical reads = 0
 Buffer pool temporary data logical reads = 0
 Buffer pool temporary data physical reads = 0
 Buffer pool index logical reads = 0
 Buffer pool index physical reads = 0
 Buffer pool temporary index logical reads = 0
 Buffer pool temporary index physical reads = 0
Total execution time (sec.ms) = 1.162904
 Total user cpu time (sec.ms) = 0.000000
 Total system cpu time (sec.ms) = 0.000000
Statement text = SELECT A0."PS_SUPPLYCOST",
 A1."S_NATIONKEY" FROM "TPCD"."PARTSUPP" A0, "TPCD"."SUPPLIER" A1 WHERE (:HO =
 A0."PS_PARTKEY") AND (A1."S_SUPPKEY" = A0."PS_SUPPKEY") FOR READ ONLY

Number of executions = 4668
 Number of compilations = 1
 Worst preparation time (ms) = 0
 Best preparation time (ms) = 0
 Internal rows deleted = 0
 Internal rows inserted = 0
Rows read = 4668

```

Internal rows updated          = 0
Rows written                   = 0
Statement sorts                = 0
Buffer pool data logical reads = 0
Buffer pool data physical reads = 0
Buffer pool temporary data logical reads = 0
Buffer pool temporary data physical reads = 0
Buffer pool index logical reads = 0
Buffer pool index physical reads = 0
Buffer pool temporary index logical reads = 0
Buffer pool temporary index physical reads = 0
Total execution time (sec.ms)    = 0.175913
Total user cpu time (sec.ms)    = 0.000000
Total system cpu time (sec.ms)  = 0.000000
Statement text                  = SELECT A0."S_NATIONKEY", A0."S_ACCTBAL",
A0."S_NAME", A0."S_ADDRESS", A0."S_PHONE", A0."S_COMMENT" FROM
"TPCD"."SUPPLIER" A0 WHERE (A0."S_SUPPKEY" = :H0 ) ORDER BY A0."S_SUPPKEY" ASC,
1 ASC FOR READ ONLY

```

As discussed in 4.2.3, “Federated server or remote data source” on page 136, since there is no direct mechanism to link the remote SQL fragments in the dynamic cache with their corresponding user SQL statement, we generated **db2exfmt** output for the user query, as shown in Example 4-50, to obtain this link.

Example 4-50 db2exfmt output for DB2_MAXIMAL_PUSHDOWN = 'N'

```

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

```

```

***** EXPLAIN INSTANCE *****

```

```

DB2_VERSION: 08.02.0
SOURCE_NAME: SQLC2E05
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-08-06-16.09.22.205228
EXPLAIN_REQUESTER: DB2I64

```

Database Context:

```

-----
Parallelism: None
CPU Speed: 5.392596e-07
Comm Speed: 100
Buffer Pool size: 75000
Sort Heap size: 100000

```

Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 640

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 203 -----

QUERYNO: 1
QUERYTAG:
Statement Type: Select
Updatable: No
Deletable: No
Query Degree: 1

Original Statement:

**SELECT S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY, P_MFGR, S_ADDRESS, S_PHONE,
S_COMMENT
FROM db2.PART, db2.SUPPLIER, db2.PARTSUPP, ora.NATION, ora.REGION
WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY AND P_SIZE = 15 AND
P_TYPE LIKE '%BRASS' AND S_NATIONKEY = N_NATIONKEY AND R_NAME =
'EUROPE' AND PS_SUPPLYCOST =
(SELECT MIN(PS_SUPPLYCOST)
FROM db2.PARTSUPP, db2.SUPPLIER, ora.NATION, ora.REGION
WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY AND S_NATIONKEY =
N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY AND R_NAME = 'EUROPE')
ORDER BY S_ACCTBAL DESC, N_NAME, S_NAME, P_PARTKEY
FETCH FIRST 100 ROWS ONLY**

Optimized Statement:

**SELECT Q10.S_ACCTBAL AS "S_ACCTBAL", Q10.S_NAME AS "S_NAME", Q8.N_NAME AS
"N_NAME", Q11.P_PARTKEY AS "P_PARTKEY", Q11.P_MFGR AS "P_MFGR",
Q10.S_ADDRESS AS "S_ADDRESS", Q10.S_PHONE AS "S_PHONE", Q10.S_COMMENT
AS "S_COMMENT"
FROM
(SELECT MIN(Q5.\$C0)
FROM
(SELECT Q4.PS_SUPPLYCOST**

```

FROM ORA.REGION AS Q1, ORA.NATION AS Q2, DB2.SUPPLIER AS Q3,
      DB2.PARTSUPP AS Q4
WHERE (Q1.R_NAME = 'EUROPE ') AND (Q2.N_REGIONKEY = Q1.R_REGIONKEY) AND
      (Q3.S_NATIONKEY = Q2.N_NATIONKEY) AND (Q3.S_SUPPKEY =
      Q4.PS_SUPPKEY) AND (Q11.P_PARTKEY = Q4.PS_PARTKEY)) AS Q5) AS
Q6, ORA.REGION AS Q7, ORA.NATION AS Q8, DB2.PARTSUPP AS Q9,
      DB2.SUPPLIER AS Q10, DB2.PART AS Q11
WHERE (Q9.PS_SUPPLYCOST = Q6.$C0) AND (Q7.R_NAME = 'EUROPE ') AND
      (Q10.S_NATIONKEY = Q8.N_NATIONKEY) AND (Q11.P_TYPE LIKE '%BRASS') AND
      (Q11.P_SIZE = 15) AND (Q10.S_SUPPKEY = Q9.PS_SUPPKEY) AND
      (Q11.P_PARTKEY = Q9.PS_PARTKEY)
ORDER BY Q10.S_ACCTBAL DESC, Q8.N_NAME, Q10.S_NAME, Q11.P_PARTKEY

```

Access Plan:

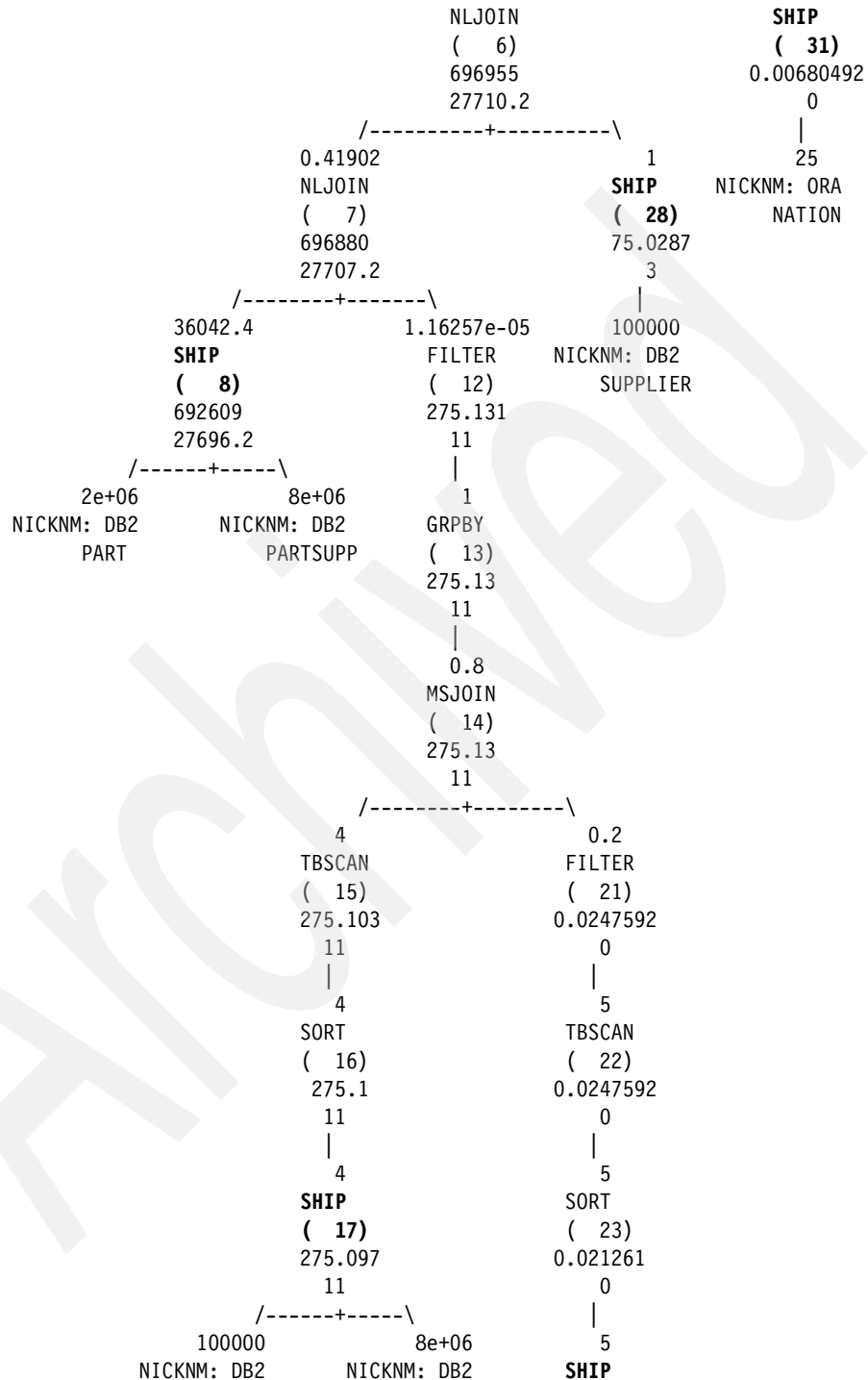
Total Cost: 696955

Query Degree:1

```

Rows
RETURN
( 1)
Cost
I/O
|
0.41902
NLJOIN
( 2)
696955
27710.2
/-----+-----\
0.41902          1
TBSCAN          SHIP
( 3)          ( 33)
696955          0.0047045
27710.2          0
|              |
0.41902          5
SORT          NICKNM: ORA
( 4)          REGION
696955
27710.2
|
0.41902
NLJOIN
( 5)
696955
27710.2
/-----+-----\
0.41902          1

```



SUPPLIER	PARTSUPP	(24)
		0.0180496
		0
		/-----+-----\
	5	25
NICKNM: ORA		NICKNM: ORA
REGION		NATION

1) RETURN: (Return Result)

Cumulative Total Cost: 696955

Cumulative CPU Cost: 7.78946e+09

Cumulative I/O Cost: 27710.2

Cumulative Re-Total Cost: 4107.72

Cumulative Re-CPU Cost: 7.61733e+09

Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 696955

Estimated Bufferpool Buffers: 1

Remote communication cost:20278.7

Arguments:

BLDLEVEL: (Build level)

DB2 v8.1.1.64 : s040509

ENVVAR : (Environment Variable)

DB2_EXTENDED_OPTIMIZATION = ON

STMTHEAP: (Statement heap size)

8192

Input Streams:

29) From Operator #2

Estimated number of rows: 0.41902

Number of columns: 8

Subquery predicate ID: Not Applicable

Column Names:

+Q12.S_ACCTBAL(D)+Q12.N_NAME(A)+Q12.S_NAME(A)

+Q12.P_PARTKEY(A)+Q12.S_COMMENT+Q12.S_PHONE

+Q12.S_ADDRESS+Q12.P_MFGR

2) NLJOIN: (Nested Loop Join)

Cumulative Total Cost: 696955

Cumulative CPU Cost: 7.78946e+09

Cumulative I/O Cost: 27710.2
Cumulative Re-Total Cost: 4107.72
Cumulative Re-CPU Cost: 7.61733e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 696955
Estimated Bufferpool Buffers: 1
Remote communication cost:20278.7

Arguments:

EARLYOUT: (Early Out flag)
 NONE
FETCHMAX: (Override for FETCH MAXPAGES)
 IGNORE
ISCANMAX: (Override for ISCAN MAXPAGES)
 IGNORE

Input Streams:

26) From Operator #3

 Estimated number of rows: 0.41902
 Number of columns: 8
 Subquery predicate ID: Not Applicable

 Column Names:

 +Q10.S_ACCTBAL(D)+Q8.N_NAME(A)+Q10.S_NAME(A)
 +Q11.P_PARTKEY(A)+Q10.S_COMMENT+Q10.S_PHONE
 +Q10.S_ADDRESS+Q11.P_MFGR

28) From Operator #33

 Estimated number of rows: 1
 Number of columns: 0
 Subquery predicate ID: Not Applicable

Output Streams:

29) To Operator #1

 Estimated number of rows: 0.41902
 Number of columns: 8
 Subquery predicate ID: Not Applicable

 Column Names:

 +Q12.S_ACCTBAL(D)+Q12.N_NAME(A)+Q12.S_NAME(A)

+Q12.P_PARTKEY(A)+Q12.S_COMMENT+Q12.S_PHONE
+Q12.S_ADDRESS+Q12.P_MFGR

3) TBSCAN: (Table Scan)

Cumulative Total Cost: 696955
Cumulative CPU Cost: 7.78945e+09
Cumulative I/O Cost: 27710.2
Cumulative Re-Total Cost: 4107.72
Cumulative Re-CPU Cost: 7.61732e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 696955
Estimated Bufferpool Buffers: 0
Remote communication cost:20272.5

Arguments:

JN INPUT: (Join input leg)
OUTER
MAXPAGES: (Maximum pages for prefetch)
ALL
PREFETCH: (Type of Prefetch)
NONE
SCANDIR : (Scan Direction)
FORWARD

Input Streams:

25) From Operator #4

Estimated number of rows: 0.41902
Number of columns: 8
Subquery predicate ID: Not Applicable

Column Names:

+Q10.S_ACCTBAL(D)+Q8.N_NAME(A)+Q10.S_NAME(A)
+Q11.P_PARTKEY(A)+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q11.P_MFGR

Output Streams:

26) To Operator #2

Estimated number of rows: 0.41902
Number of columns: 8
Subquery predicate ID: Not Applicable

Column Names:

+Q10.S_ACCTBAL(D)+Q8.N_NAME(A)+Q10.S_NAME(A)
+Q11.P_PARTKEY(A)+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q11.P_MFGR

4) SORT : (Sort)

Cumulative Total Cost: 696955
Cumulative CPU Cost: 7.78945e+09
Cumulative I/O Cost: 27710.2
Cumulative Re-Total Cost: 4107.71
Cumulative Re-CPU Cost: 7.61732e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 696955
Estimated Bufferpool Buffers: 27701.2
Remote communication cost:20272.5

Arguments:

DUPLWARN: (Duplicates Warning flag)
FALSE
NUMROWS : (Estimated number of rows)
1
ROWWIDTH: (Estimated width of rows)
204
SORTKEY : (Sort Key column)
1: Q10.S_ACCTBAL(D)
SORTKEY : (Sort Key column)
2: Q8.N_NAME(A)
SORTKEY : (Sort Key column)
3: Q10.S_NAME(A)
SORTKEY : (Sort Key column)
4: Q11.P_PARTKEY(A)
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

24) From Operator #5

Estimated number of rows: 0.41902
Number of columns: 12
Subquery predicate ID: Not Applicable

Column Names:

```

+Q6.$C0+Q8.N_NAME+Q9.PS_SUPPLYCOST
+Q9.PS_SUPPKEY+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q10.S_NAME+Q10.S_ACCTBAL
+Q10.S_NATIONKEY+Q11.P_MFGR+Q11.P_PARTKEY

```

Output Streams:

25) To Operator #3

```

Estimated number of rows: 0.41902
Number of columns: 8
Subquery predicate ID: Not Applicable

```

Column Names:

```

+Q10.S_ACCTBAL(D)+Q8.N_NAME(A)+Q10.S_NAME(A)
+Q11.P_PARTKEY(A)+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q11.P_MFGR

```

5) NLJOIN: (Nested Loop Join)

```

Cumulative Total Cost: 696955
Cumulative CPU Cost: 7.78945e+09
Cumulative I/O Cost: 27710.2
Cumulative Re-Total Cost: 4107.71
Cumulative Re-CPU Cost: 7.61732e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 696955
Estimated Bufferpool Buffers: 27701.2
Remote communication cost:20272.5

```

Arguments:

```

EARLYOUT: (Early Out flag)
  NONE
FETCHMAX: (Override for FETCH MAXPAGES)
  IGNORE
ISCANMAX: (Override for ISCAN MAXPAGES)
  IGNORE

```

Predicates:

```

4) Predicate used in Join
  Relational Operator: Equal (=)
  Subquery Input Required: No
  Filter Factor: 0.04

```

Predicate Text:

(Q10.S_NATIONKEY = Q8.N_NATIONKEY)

Input Streams:

21) From Operator #6

Estimated number of rows: 0.41902
Number of columns: 11
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0+Q9.PS_SUPPLYCOST+Q9.PS_SUPPKEY
+Q10.S_COMMENT+Q10.S_PHONE+Q10.S_ADDRESS
+Q10.S_NAME+Q10.S_ACCTBAL+Q10.S_NATIONKEY
+Q11.P_MFGR+Q11.P_PARTKEY

23) From Operator #31

Estimated number of rows: 1
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q8.N_NAME

Output Streams:

24) To Operator #4

Estimated number of rows: 0.41902
Number of columns: 12
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0+Q8.N_NAME+Q9.PS_SUPPLYCOST
+Q9.PS_SUPPKEY+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q10.S_NAME+Q10.S_ACCTBAL
+Q10.S_NATIONKEY+Q11.P_MFGR+Q11.P_PARTKEY

6) NLJOIN: (Nested Loop Join)
Cumulative Total Cost: 696955
Cumulative CPU Cost: 7.78943e+09

Cumulative I/O Cost: 27710.2
Cumulative Re-Total Cost: 4107.71
Cumulative Re-CPU Cost: 7.61731e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 696955
Estimated Bufferpool Buffers: 27700.2
Remote communication cost:20266.3

Arguments:

EARLYOUT: (Early Out flag)
 NONE
FETCHMAX: (Override for FETCH MAXPAGES)
 IGNORE
ISCANMAX: (Override for ISCAN MAXPAGES)
 IGNORE
JN INPUT: (Join input leg)
 OUTER

Predicates:

7) Predicate used in Join
 Relational Operator: Equal (=)
 Subquery Input Required: No
 Filter Factor: 1e-05

Predicate Text:

(Q10.S_SUPPKEY = Q9.PS_SUPPKEY)

Input Streams:

18) From Operator #7

 Estimated number of rows: 0.41902
 Number of columns: 5
 Subquery predicate ID: Not Applicable

 Column Names:

 +Q6.\$C0+Q9.PS_SUPPLYCOST+Q9.PS_SUPPKEY
 +Q11.P_MFGR+Q11.P_PARTKEY

20) From Operator #28

 Estimated number of rows: 1
 Number of columns: 7
 Subquery predicate ID: Not Applicable

Column Names:

+Q10.S_SUPPKEY(A)+Q10.S_NATIONKEY(A)
+Q10.S_COMMENT+Q10.S_PHONE+Q10.S_ADDRESS
+Q10.S_NAME+Q10.S_ACCTBAL

Output Streams:

21) To Operator #5

Estimated number of rows: 0.41902
Number of columns: 11
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0+Q9.PS_SUPPLYCOST+Q9.PS_SUPPKEY
+Q10.S_COMMENT+Q10.S_PHONE+Q10.S_ADDRESS
+Q10.S_NAME+Q10.S_ACCTBAL+Q10.S_NATIONKEY
+Q11.P_MFGR+Q11.P_PARTKEY

7) NLJOIN: (Nested Loop Join)

Cumulative Total Cost: 696880
Cumulative CPU Cost: 7.78938e+09
Cumulative I/O Cost: 27707.2
Cumulative Re-Total Cost: 4107.69
Cumulative Re-CPU Cost: 7.61728e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 696880
Estimated Bufferpool Buffers: 27697.2
Remote communication cost:20260.1

Arguments:

EARLYOUT: (Early Out flag)
NONE
FETCHMAX: (Override for FETCH MAXPAGES)
IGNORE
ISCANMAX: (Override for ISCAN MAXPAGES)
IGNORE
JN INPUT: (Join input leg)
OUTER

Predicates:

2) Predicate used in Join

Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 1.16257e-05

Predicate Text:

(Q9.PS_SUPPLYCOST = Q6.\$C0)

Input Streams:

3) From Operator #8

Estimated number of rows: 36042.4
Number of columns: 4
Subquery predicate ID: Not Applicable

Column Names:

+Q9.PS_SUPPLYCOST+Q9.PS_SUPPKEY+Q11.P_MFGR
+Q11.P_PARTKEY

17) From Operator #12

Estimated number of rows: 1.16257e-05
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0

Output Streams:

18) To Operator #6

Estimated number of rows: 0.41902
Number of columns: 5
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0+Q9.PS_SUPPLYCOST+Q9.PS_SUPPKEY
+Q11.P_MFGR+Q11.P_PARTKEY

8) SHIP : (Ship)

Cumulative Total Cost: 692609

Cumulative CPU Cost: 3.79842e+08
Cumulative I/O Cost: 27696.2
Cumulative Re-Total Cost: 112.05
Cumulative Re-CPU Cost: 2.07785e+08
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 151.483
Estimated Bufferpool Buffers: 27697.2
Remote communication cost:20237.9

Arguments:

CSERQY : (Remote common subexpression)
FALSE

DSTSEVER: (Destination (ship to) server)
- (NULL).

JN INPUT: (Join input leg)
OUTER

RMTQTX : (Remote statement)

SELECT A0."P_PARTKEY", A0."P_MFGR", A1."PS_SUPPKEY",
A1."PS_SUPPLYCOST" FROM "TPCD"."PART" A0, "TPCD"."PARTSUPP" A1 WHERE
(A0."P_SIZE" = 15) AND (A0."P_TYPE" LIKE '%BRASS') AND (A0."P_PARTKEY" =
A1."PS_PARTKEY") FOR READ ONLY

SRCSEVER: (Source (ship from) server)
DB2SERV

STREAM : (Remote stream)
FALSE

Input Streams:

1) From Object DB2.PART

Estimated number of rows: 2e+06
Number of columns: 5
Subquery predicate ID: Not Applicable

Column Names:

+Q11.\$RID\$+Q11.P_MFGR+Q11.P_TYPE+Q11.P_SIZE
+Q11.P_PARTKEY

2) From Object DB2.PARTSUPP

Estimated number of rows: 8e+06
Number of columns: 4
Subquery predicate ID: Not Applicable

Column Names:

+Q9.\$RID\$+Q9.PS_SUPPLYCOST+Q9.PS_SUPPKEY

+Q9.PS_PARTKEY

Output Streams:

3) To Operator #7

Estimated number of rows: 36042.4

Number of columns: 4

Subquery predicate ID: Not Applicable

Column Names:

+Q9.PS_SUPPLYCOST+Q9.PS_SUPPKEY+Q11.P_MFGR

+Q11.P_PARTKEY

12) FILTER: (Filter)

Cumulative Total Cost: 275.131

Cumulative CPU Cost: 242977

Cumulative I/O Cost: 11

Cumulative Re-Total Cost: 275.131

Cumulative Re-CPU Cost: 242977

Cumulative Re-I/O Cost: 11

Cumulative First Row Cost: 275.131

Estimated Bufferpool Buffers: 0

Remote communication cost:22.2188

Arguments:

JN INPUT: (Join input leg)
INNER

Predicates:

2) Residual Predicate

Relational Operator: Equal (=)

Subquery Input Required: No

Filter Factor: 1.16257e-05

Predicate Text:

(Q9.PS_SUPPLYCOST = Q6.\$C0)

Input Streams:

16) From Operator #13

Estimated number of rows: 1
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0

Output Streams:

17) To Operator #7

Estimated number of rows: 1.16257e-05
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0

13) GRPBY : (Group By)
Cumulative Total Cost: 275.13
Cumulative CPU Cost: 241642
Cumulative I/O Cost: 11
Cumulative Re-Total Cost: 275.13
Cumulative Re-CPU Cost: 241642
Cumulative Re-I/O Cost: 11
Cumulative First Row Cost: 275.13
Estimated Bufferpool Buffers: 0
Remote communication cost:22.2188

Arguments:

AGGMODE : (Aggregation Mode)
COMPLETE
GROUPBYC: (Group By columns)
FALSE
GROUPBYN: (Number of Group By columns)
0
ONEFETCH: (One Fetch flag)
FALSE

Input Streams:

15) From Operator #14

Estimated number of rows: 0.8

Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q5.PS_SUPPLYCOST

Output Streams:

16) To Operator #12

Estimated number of rows: 1
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0

14) MSJOIN: (Merge Scan Join)

Cumulative Total Cost: 275.13
Cumulative CPU Cost: 241192
Cumulative I/O Cost: 11
Cumulative Re-Total Cost: 275.13
Cumulative Re-CPU Cost: 241192
Cumulative Re-I/O Cost: 11
Cumulative First Row Cost: 275.13
Estimated Bufferpool Buffers: 0
Remote communication cost:22.2188

Arguments:

EARLYOUT: (Early Out flag)
NONE
INNERCOL: (Inner Order Columns)
1: Q2.N_NATIONKEY(A)
OUTERCOL: (Outer Order columns)
1: Q3.S_NATIONKEY(A)
TEMPSIZE: (Temporary Table Page Size)
4096

Predicates:

11) Predicate used in Join
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 0.04

Predicate Text:

(Q3.S_NATIONKEY = Q2.N_NATIONKEY)

Input Streams:

8) From Operator #15

Estimated number of rows: 4

Number of columns: 2

Subquery predicate ID: Not Applicable

Column Names:

+Q3.S_NATIONKEY(A)+Q4.PS_SUPPLYCOST

14) From Operator #21

Estimated number of rows: 0.2

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

Output Streams:

15) To Operator #13

Estimated number of rows: 0.8

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q5.PS_SUPPLYCOST

15) TBSCAN: (Table Scan)

Cumulative Total Cost: 275.103

Cumulative CPU Cost: 191030

Cumulative I/O Cost: 11

Cumulative Re-Total Cost: 25.0566

Cumulative Re-CPU Cost: 104891

Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 275.101
Estimated Bufferpool Buffers: 0
Remote communication cost:10.8594

Arguments:

JN INPUT: (Join input leg)
OUTER
MAXPAGES: (Maximum pages for prefetch)
ALL
PREFETCH: (Type of Prefetch)
NONE
SCANDIR : (Scan Direction)
FORWARD

Input Streams:

7) From Operator #16

Estimated number of rows: 4
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q3.S_NATIONKEY(A)+Q4.PS_SUPPLYCOST

Output Streams:

8) To Operator #14

Estimated number of rows: 4
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q3.S_NATIONKEY(A)+Q4.PS_SUPPLYCOST

16) SORT : (Sort)
Cumulative Total Cost: 275.1
Cumulative CPU Cost: 185333
Cumulative I/O Cost: 11
Cumulative Re-Total Cost: 25.0535
Cumulative Re-CPU Cost: 99194
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 275.1

Estimated Bufferpool Buffers: 50012
Remote communication cost:10.8594

Arguments:

DUPLWARN: (Duplicates Warning flag)
FALSE
NUMROWS : (Estimated number of rows)
4
ROWWIDTH: (Estimated width of rows)
16
SORTKEY : (Sort Key column)
1: Q3.S_NATIONKEY(A)
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

6) From Operator #17

Estimated number of rows: 4
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q3.S_NATIONKEY+Q4.PS_SUPPLYCOST

Output Streams:

7) To Operator #15

Estimated number of rows: 4
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q3.S_NATIONKEY(A)+Q4.PS_SUPPLYCOST

17) SHIP : (Ship)

Cumulative Total Cost: 275.097
Cumulative CPU Cost: 180037
Cumulative I/O Cost: 11
Cumulative Re-Total Cost: 25.0535

Cumulative Re-CPU Cost: 99194
Cumulative Re-I/O Cost: 1
Cumulative First Row Cost: 125.05
Estimated Bufferpool Buffers: 50012
Remote communication cost:10.8594

Arguments:

CSERQY : (Remote common subexpression)

FALSE

DSTSEVER: (Destination (ship to) server)

- (NULL).

RMTQTX : (Remote statement)

**SELECT A0."PS_SUPPLYCOST", A1."S_NATIONKEY" FROM "TPCD"."PARTSUPP"
A0, "TPCD"."SUPPLIER" A1 WHERE (:HO = A0."PS_PARTKEY") AND (A1."S_SUPPKEY" =
A0."PS_SUPPKEY") FOR READ ONLY**

SRCSEVER: (Source (ship from) server)

DB2SERV

STREAM : (Remote stream)

FALSE

Input Streams:

4) From Object DB2.PARTSUPP

Estimated number of rows: 8e+06

Number of columns: 4

Subquery predicate ID: Not Applicable

Column Names:

+Q4.\$RID\$+Q4.PS_SUPPLYCOST+Q4.PS_SUPPKEY

+Q4.PS_PARTKEY

5) From Object DB2.SUPPLIER

Estimated number of rows: 100000

Number of columns: 3

Subquery predicate ID: Not Applicable

Column Names:

+Q3.\$RID\$+Q3.S_NATIONKEY+Q3.S_SUPPKEY

Output Streams:

6) To Operator #16

Estimated number of rows: 4
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q3.S_NATIONKEY+Q4.PS_SUPPLYCOST

21) FILTER: (Filter)

Cumulative Total Cost: 0.0247592
Cumulative CPU Cost: 45913.4
Cumulative I/O Cost: 0
Cumulative Re-Total Cost: 0.0103888
Cumulative Re-CPU Cost: 19265
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 0.022036
Estimated Bufferpool Buffers: 0
Remote communication cost:11.3594

Arguments:

JN INPUT: (Join input leg)
INNER

Predicates:

11) Residual Predicate
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 0.04

Predicate Text:

(Q3.S_NATIONKEY = Q2.N_NATIONKEY)

Input Streams:

13) From Operator #22

Estimated number of rows: 5
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

Output Streams:

14) To Operator #14

Estimated number of rows: 0.2

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

22) TBSCAN: (Table Scan)

Cumulative Total Cost: 0.0247592

Cumulative CPU Cost: 45913.4

Cumulative I/O Cost: 0

Cumulative Re-Total Cost: 0.0103888

Cumulative Re-CPU Cost: 19265

Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 0.022036

Estimated Bufferpool Buffers: 0

Remote communication cost:11.3594

Arguments:

MAXPAGES: (Maximum pages for prefetch)

ALL

PREFETCH: (Type of Prefetch)

NONE

SCANDIR : (Scan Direction)

FORWARD

Input Streams:

12) From Operator #23

Estimated number of rows: 5

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

Output Streams:

13) To Operator #21

Estimated number of rows: 5
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

23) SORT : (Sort)

Cumulative Total Cost: 0.021261
Cumulative CPU Cost: 39426.4
Cumulative I/O Cost: 0
Cumulative Re-Total Cost: 0.00689066
Cumulative Re-CPU Cost: 12778
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 0.021261
Estimated Bufferpool Buffers: 2
Remote communication cost:11.3594

Arguments:

DUPLWARN: (Duplicates Warning flag)
FALSE
NUMROWS : (Estimated number of rows)
5
ROWWIDTH: (Estimated width of rows)
12
SORTKEY : (Sort Key column)
1: Q2.N_NATIONKEY(A)
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

11) From Operator #24

Estimated number of rows: 5
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY

Output Streams:

12) To Operator #22

Estimated number of rows: 5

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

24) SHIP : (Ship)

Cumulative Total Cost: 0.0180496

Cumulative CPU Cost: 33471

Cumulative I/O Cost: 0

Cumulative Re-Total Cost: 0.00689066

Cumulative Re-CPU Cost: 12778

Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 0.0103301

Estimated Bufferpool Buffers: 2

Remote communication cost:11.3594

Arguments:

CSERQY : (Remote common subexpression)

FALSE

DSTSEVER: (Destination (ship to) server)

- (NULL).

RMTQTX : (Remote statement)

SELECT A1."N_NATIONKEY" FROM "IITEST"."REGION" A0, "IITEST"."NATION"
A1 WHERE (A0."R_NAME" = 'EUROPE ') AND (A1."N_REGIONKEY" =
A0."R_REGIONKEY")

SRCSEVER: (Source (ship from) server)

ORASERV

STREAM : (Remote stream)

FALSE

Input Streams:

9) From Object ORA.REGION

Estimated number of rows: 5

Number of columns: 3

Subquery predicate ID: Not Applicable

Column Names:

```
-----  
+Q1.$RID$+Q1.R_NAME+Q1.R_REGIONKEY
```

10) From Object ORA.NATION

```
Estimated number of rows: 25  
Number of columns: 3  
Subquery predicate ID: Not Applicable
```

```
Column Names:  
-----  
+Q2.$RID$+Q2.N_REGIONKEY+Q2.N_NATIONKEY
```

Output Streams:

11) To Operator #23

```
Estimated number of rows: 5  
Number of columns: 1  
Subquery predicate ID: Not Applicable
```

```
Column Names:  
-----  
+Q2.N_NATIONKEY
```

28) SHIP : (Ship)

```
Cumulative Total Cost: 75.0287  
Cumulative CPU Cost: 53193.6  
Cumulative I/O Cost: 3  
Cumulative Re-Total Cost: 50.021  
Cumulative Re-CPU Cost: 38888.6  
Cumulative Re-I/O Cost: 2  
Cumulative First Row Cost: 75.0271  
Estimated Bufferpool Buffers: 4520  
Remote communication cost:9.35938
```

Arguments:

```
-----  
CSERQY : (Remote common subexpression)  
FALSE  
DSTSEVER: (Destination (ship to) server)  
- (NULL).  
JN INPUT: (Join input leg)  
INNER  
RMTQTX : (Remote statement)
```

```

        SELECT AO."S_NATIONKEY", AO."S_ACCTBAL", AO."S_NAME", AO."S_ADDRESS",
        AO."S_PHONE", AO."S_COMMENT" FROM "TPCD"."SUPPLIER" AO WHERE (AO."S_SUPPKEY" =
:HO ) ORDER BY AO."S_SUPPKEY" ASC, 1 ASC FOR READ ONLY

```

SRCSEVER: (Source (ship from) server)

DB2SERV

STREAM : (Remote stream)

FALSE

Input Streams:

19) From Object DB2.SUPPLIER

Estimated number of rows: 100000

Number of columns: 5

Subquery predicate ID: Not Applicable

Column Names:

+Q10.S_COMMENT+Q10.S_PHONE+Q10.S_ADDRESS

+Q10.S_NAME+Q10.S_ACCTBAL

Output Streams:

20) To Operator #6

Estimated number of rows: 1

Number of columns: 7

Subquery predicate ID: Not Applicable

Column Names:

+Q10.S_SUPPKEY(A)+Q10.S_NATIONKEY(A)

+Q10.S_COMMENT+Q10.S_PHONE+Q10.S_ADDRESS

+Q10.S_NAME+Q10.S_ACCTBAL

31) SHIP : (Ship)

Cumulative Total Cost: 0.00680492

Cumulative CPU Cost: 12619

Cumulative I/O Cost: 0

Cumulative Re-Total Cost: 0.00448071

Cumulative Re-CPU Cost: 8309

Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 0.0055959

Estimated Bufferpool Buffers: 1

Remote communication cost:9.35938

Arguments:

```

-----
CSERQY : (Remote common subexpression)
  FALSE
DSTSEVER: (Destination (ship to) server)
  - (NULL).
JN INPUT: (Join input leg)
  INNER
RMTQTXT : (Remote statement)
  SELECT AO."N_NAME" FROM "IITEST"."NATION" AO WHERE (:HO =
AO."N_NATIONKEY")
SRCSEVER: (Source (ship from) server)
  ORASERV
STREAM : (Remote stream)
  FALSE

```

Input Streams:

```

-----
22) From Object ORA.NATION

```

```

    Estimated number of rows: 25
    Number of columns: 3
    Subquery predicate ID: Not Applicable

```

Column Names:

```

-----
+Q8.$RID$+Q8.N_NAME+Q8.N_NATIONKEY

```

Output Streams:

```

-----
23) To Operator #5

```

```

    Estimated number of rows: 1
    Number of columns: 1
    Subquery predicate ID: Not Applicable

```

Column Names:

```

-----
+Q8.N_NAME

```

33) SHIP : (Ship)

```

Cumulative Total Cost: 0.0047045
Cumulative CPU Cost: 8724
Cumulative I/O Cost: 0
Cumulative Re-Total Cost: 0.00240995
Cumulative Re-CPU Cost: 4469
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 0.00352514

```

Estimated Bufferpool Buffers: 1
Remote communication cost:9.35938

Arguments:

CSERQY : (Remote common subexpression)
FALSE
DSTSEVER: (Destination (ship to) server)
- (NULL).
JN INPUT: (Join input leg)
INNER
RMTQTX : (Remote statement)
SELECT '1' FROM "IITEST"."REGION" AO WHERE (AO."R_NAME" = 'EUROPE
)
SRCSEVER: (Source (ship from) server)
ORASERV
STREAM : (Remote stream)
FALSE

Input Streams:

27) From Object ORA.REGION

Estimated number of rows: 5
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q7.\$RID\$+Q7.R_NAME

Output Streams:

28) To Operator #2

Estimated number of rows: 1
Number of columns: 0
Subquery predicate ID: Not Applicable

Objects Used in Access Plan:

Schema: DB2I64
Name: ORDERSMQT
Type: Materialized View (reference only)

Schema: DB2

Name: PART
Type: Nickname
Time of creation: 2004-06-17-10.42.49.959240
Last statistics update:
Number of columns: 9
Number of rows: 2000000
Width of rows: 70
Number of buffer pool pages: 76238
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: DB2
Name: PARTSUPP
Type: Nickname
Time of creation: 2004-06-17-10.42.55.831083
Last statistics update:
Number of columns: 5
Number of rows: 8000000
Width of rows: 28
Number of buffer pool pages: 319290
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: DB2
Name: SUPPLIER
Type: Nickname
Time of creation: 2004-06-17-10.42.53.722466
Last statistics update:
Number of columns: 7
Number of rows: 100000
Width of rows: 157
Number of buffer pool pages: 4093
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: ORA
Name: NATION
Type: Nickname
Time of creation: 2004-06-16-23.22.43.109494
Last statistics update:
Number of columns: 4
Number of rows: 25
Width of rows: 62
Number of buffer pool pages: 15
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: ORA
Name: REGION
Type: Nickname
Time of creation: 2004-06-16-23.22.43.342734
Last statistics update:
Number of columns: 3
Number of rows: 5
Width of rows: 56
Number of buffer pool pages: 15
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

There are six SHIP operators (8, 17, 24, 28, 31 and 33) in which the RMTQTX fields identify the user query's corresponding remote SQL fragments. These are highlighted in SHIP operator 5 in Example 4-50 on page 278.

These remote SQL fragments are located in the dynamic cache output shown in Example 4-49 on page 274, and the following information can be gathered:

- **Number of executions** is 1 for the user-entered query and varying number of executions for each of the six remote SQL fragments.
- **Total execution time (sec.ms)**, which is 80.444751 seconds for the user-entered query, and varying times (all of which are fractions of seconds per execution) for each of the six remote SQL fragments.

- **Total user cpu time (sec.ms)** and **Total system cpu time (sec.ms)**, which are 9.920000 and 1.980000 for the user-entered query, and zero for the remote SQL fragments.
- **Rows read** is 100 for the user-entered query, which is the number of rows returned to the user, and varying number of rows for each of the remote SQL fragments, which indicates the number of rows returned to the federated server from the particular remote data source.
- Other fields of interest include **Statement sorts**, **Statement sort overflows**, and **Total sort time**, which only apply to the user-entered query, and have values 62833, 1, and 13 milliseconds, respectively.
- The buffer pool hit ratio appears to be 100 percent.

Note: To obtain the average elapsed and CPU times, as well as the number of rows returned, you must divide the values shown by the **Number of executions**.

We can derive the following information from these metrics: The average elapsed time for the user query is $(80.444751 / 1) = 130.732961$ seconds, while that of the remote SQL fragments are fractions of seconds.

Attention: In our example, it is clear that the predominant portion of the query time is at the federated server.

This performance problem therefore needs to be investigated at the federated server only, as described in 4.2.4, “Federated server related” on page 152.

► **Federated server related.**

As discussed in 4.2.4, “Federated server related” on page 152, poor performance at the federated server may be caused by a number of factors such as statistics, index information, pushdown, joins, and parallelism.

Each of these factors need to be evaluated in turn to determine the root cause of the performance problem.

- A review of the number of rows estimated by the DB2 optimizer in Example 4-50 on page 278 as being returned from each of the remote data sources to the federated server, and the actual number of rows returned (**Rows read** field) from each of the remote data sources in Example 4-49 on page 274 show discrepancies that merit investigation. Since this process has already been discussed in 4.4.2, “Missing or incorrect statistics/index information” on page 170, we will not repeat it in this scenario.

- A review of the remote SQL fragments text and the original statement, as shown in Example 4-50 on page 278, show that there are predicates that need to be investigated from a pushdown perspective.

Some of the join predicates need to be executed at the federated server because they join nicknames across different data sources, but it appears that some joins between tables at the remote DB2 data source have not been pushed down. To determine if this was the case, we need to set DB2_MAXIMAL_PUSHDOWN to 'Y' (if it was set to the default of 'N'), as shown in Example 4-51, and regenerate **db2exfmt** output.

Example 4-51 Alter the wrapper option DB2_MAXIMAL_PUSHDOWN to 'Y'

```
connect to fedserv
```

Database Connection Information

```
Database server      = DB2/AIX64 8.2.0
SQL authorization ID = DB2I64P
Local database alias = FEDSERV
```

```
alter wrapper DRDA options (set DB2_MAXIMAL_PUSHDOWN 'Y')
DB20000I The SQL command completed successfully.
```

Note: The server option may be also be set temporarily using the SET SERVER OPTION.

- **db2exfmt** output corresponding to DB2_MAXIMAL_PUSHDOWN = 'Y' is shown in Example 4-52.

Example 4-52 db2exfmt output for DB2_MAXIMAL_PUSHDOWN = 'Y'

```
DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool
```

```
***** EXPLAIN INSTANCE *****
```

```
DB2_VERSION: 08.02.0
SOURCE_NAME: SQLC2E05
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-08-06-16.16.43.437095
EXPLAIN_REQUESTER: DB2I64
```

Database Context:

Parallelism: None
CPU Speed: 5.392596e-07
Comm Speed: 100
Buffer Pool size: 75000
Sort Heap size: 100000
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 640

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 203 -----

QUERYNO: 1
QUERYTAG:
Statement Type: Select
Updatable: No
Deletable: No
Query Degree: 1

Original Statement:

SELECT S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY, P_MFGR, S_ADDRESS, S_PHONE,
S_COMMENT
FROM db2.PART, db2.SUPPLIER, db2.PARTSUPP, ora.NATION, ora.REGION
WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY AND P_SIZE = 15 AND
P_TYPE LIKE '%BRASS' AND S_NATIONKEY = N_NATIONKEY AND R_NAME =
'EUROPE' AND PS_SUPPLYCOST =
(SELECT MIN(PS_SUPPLYCOST)
FROM db2.PARTSUPP, db2.SUPPLIER, ora.NATION, ora.REGION
WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY AND S_NATIONKEY =
N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY AND R_NAME = 'EUROPE')
ORDER BY S_ACCTBAL DESC, N_NAME, S_NAME, P_PARTKEY
FETCH FIRST 100 ROWS ONLY

Optimized Statement:

SELECT Q10.S_ACCTBAL AS "S_ACCTBAL", Q10.S_NAME AS "S_NAME", Q8.N_NAME AS

```

        "N_NAME", Q11.P_PARTKEY AS "P_PARTKEY", Q11.P_MFGR AS "P_MFGR",
        Q10.S_ADDRESS AS "S_ADDRESS", Q10.S_PHONE AS "S_PHONE", Q10.S_COMMENT
        AS "S_COMMENT"
FROM
    (SELECT MIN(Q5.$C0)
    FROM
        (SELECT Q4.PS_SUPPLYCOST
        FROM ORA.REGION AS Q1, ORA.NATION AS Q2, DB2.SUPPLIER AS Q3,
            DB2.PARTSUPP AS Q4
        WHERE (Q1.R_NAME = 'EUROPE ') AND (Q2.N_REGIONKEY = Q1.R_REGIONKEY) AND
            (Q3.S_NATIONKEY = Q2.N_NATIONKEY) AND (Q3.S_SUPPKEY =
            Q4.PS_SUPPKEY) AND (Q11.P_PARTKEY = Q4.PS_PARTKEY)) AS Q5) AS
        Q6, ORA.REGION AS Q7, ORA.NATION AS Q8, DB2.PARTSUPP AS Q9,
        DB2.SUPPLIER AS Q10, DB2.PART AS Q11
WHERE (Q9.PS_SUPPLYCOST = Q6.$C0) AND (Q7.R_NAME = 'EUROPE ') AND
    (Q10.S_NATIONKEY = Q8.N_NATIONKEY) AND (Q11.P_TYPE LIKE '%BRASS') AND
    (Q11.P_SIZE = 15) AND (Q10.S_SUPPKEY = Q9.PS_SUPPKEY) AND
    (Q11.P_PARTKEY = Q9.PS_PARTKEY)
ORDER BY Q10.S_ACCTBAL DESC, Q8.N_NAME, Q10.S_NAME, Q11.P_PARTKEY

```

Access Plan:

Total Cost: 713016

Query Degree:1

```

      Rows
      RETURN
      ( 1)
      Cost
      I/O
      |
      0.41902
      NLJOIN
      ( 2)
      713016
      31800.2
      /-----\
0.41902      1
TBSCAN      SHIP
( 3)      ( 31)
713016      0.0047045
31800.2      0
|      |
0.41902      5
SORT      NICKNM: ORA
( 4)      REGION
713016
31800.2
|

```


	NICKNM: DB2 SUPPLIER	NICKNM: DB2 PARTSUPP	SHIP (25) 0.0180496 0 /-----+-----\ 5
25		NICKNM: ORA	NICKNM:
ORA		REGION	
NATION			

1) RETURN: (Return Result)

Cumulative Total Cost: 713016
Cumulative CPU Cost: 8.00039e+09
Cumulative I/O Cost: 31800.2
Cumulative Re-Total Cost: 712741
Cumulative Re-CPU Cost: 8.00034e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 713016
Estimated Bufferpool Buffers: 1
Remote communication cost:26483.8

Arguments:

BLDLEVEL: (Build level)
DB2 v8.1.1.64 : s040509
ENVVAR : (Environment Variable)
DB2_EXTENDED_OPTIMIZATION = ON
STMTHEAP: (Statement heap size)
8192

Input Streams:

27) From Operator #2

Estimated number of rows: 0.41902
Number of columns: 8
Subquery predicate ID: Not Applicable

Column Names:

+Q12.S_ACCTBAL(D)+Q12.N_NAME(A)+Q12.S_NAME(A)
+Q12.P_PARTKEY(A)+Q12.S_COMMENT+Q12.S_PHONE
+Q12.S_ADDRESS+Q12.P_MFGR

2) NLJOIN: (Nested Loop Join)
Cumulative Total Cost: 713016
Cumulative CPU Cost: 8.00039e+09
Cumulative I/O Cost: 31800.2
Cumulative Re-Total Cost: 712741
Cumulative Re-CPU Cost: 8.00034e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 713016
Estimated Bufferpool Buffers: 1
Remote communication cost:26483.8

Arguments:

EARLYOUT: (Early Out flag)
NONE
FETCHMAX: (Override for FETCH MAXPAGES)
IGNORE
ISCANMAX: (Override for ISCAN MAXPAGES)
IGNORE

Input Streams:

24) From Operator #3

Estimated number of rows: 0.41902
Number of columns: 8
Subquery predicate ID: Not Applicable

Column Names:

+Q10.S_ACCTBAL(D)+Q8.N_NAME(A)+Q10.S_NAME(A)
+Q11.P_PARTKEY(A)+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q11.P_MFGR

26) From Operator #31

Estimated number of rows: 1
Number of columns: 0
Subquery predicate ID: Not Applicable

Output Streams:

27) To Operator #1

Estimated number of rows: 0.41902
Number of columns: 8
Subquery predicate ID: Not Applicable

Column Names:

+Q12.S_ACCTBAL(D)+Q12.N_NAME(A)+Q12.S_NAME(A)
+Q12.P_PARTKEY(A)+Q12.S_COMMENT+Q12.S_PHONE
+Q12.S_ADDRESS+Q12.P_MFGR

3) TBSCAN: (Table Scan)

Cumulative Total Cost: 713016
Cumulative CPU Cost: 8.00038e+09
Cumulative I/O Cost: 31800.2
Cumulative Re-Total Cost: 712741
Cumulative Re-CPU Cost: 8.00033e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 713016
Estimated Bufferpool Buffers: 0
Remote communication cost:26477.6

Arguments:

JN INPUT: (Join input leg)
OUTER
MAXPAGES: (Maximum pages for prefetch)
ALL
PREFETCH: (Type of Prefetch)
NONE
SCANDIR : (Scan Direction)
FORWARD

Input Streams:

23) From Operator #4

Estimated number of rows: 0.41902
Number of columns: 8
Subquery predicate ID: Not Applicable

Column Names:

+Q10.S_ACCTBAL(D)+Q8.N_NAME(A)+Q10.S_NAME(A)
+Q11.P_PARTKEY(A)+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q11.P_MFGR

Output Streams:

24) To Operator #2

Estimated number of rows: 0.41902
Number of columns: 8
Subquery predicate ID: Not Applicable

Column Names:

+Q10.S_ACCTBAL(D)+Q8.N_NAME(A)+Q10.S_NAME(A)
+Q11.P_PARTKEY(A)+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q11.P_MFGR

4) SORT : (Sort)
Cumulative Total Cost: 713016
Cumulative CPU Cost: 8.00038e+09
Cumulative I/O Cost: 31800.2
Cumulative Re-Total Cost: 712741
Cumulative Re-CPU Cost: 8.00033e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 713016
Estimated Bufferpool Buffers: 31791.2
Remote communication cost:26477.6

Arguments:

DUPLWARN: (Duplicates Warning flag)
FALSE
NUMROWS : (Estimated number of rows)
1
ROWWIDTH: (Estimated width of rows)
204
SORTKEY : (Sort Key column)
1: Q10.S_ACCTBAL(D)
SORTKEY : (Sort Key column)
2: Q8.N_NAME(A)
SORTKEY : (Sort Key column)
3: Q10.S_NAME(A)
SORTKEY : (Sort Key column)
4: Q11.P_PARTKEY(A)
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

22) From Operator #5

Estimated number of rows: 0.41902
Number of columns: 11

Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0+Q8.N_NAME+Q9.PS_SUPPLYCOST
+Q10.S_COMMENT+Q10.S_PHONE+Q10.S_ADDRESS
+Q10.S_NAME+Q10.S_ACCTBAL+Q10.S_NATIONKEY
+Q11.P_MFGR+Q11.P_PARTKEY

Output Streams:

23) To Operator #3

Estimated number of rows: 0.41902
Number of columns: 8
Subquery predicate ID: Not Applicable

Column Names:

+Q10.S_ACCTBAL(D)+Q8.N_NAME(A)+Q10.S_NAME(A)
+Q11.P_PARTKEY(A)+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q11.P_MFGR

5) NLJOIN: (Nested Loop Join)
Cumulative Total Cost: 713016
Cumulative CPU Cost: 8.00038e+09
Cumulative I/O Cost: 31800.2
Cumulative Re-Total Cost: 712741
Cumulative Re-CPU Cost: 8.00033e+09
Cumulative Re-I/O Cost: 31789.2
Cumulative First Row Cost: 713016
Estimated Bufferpool Buffers: 31791.2
Remote communication cost:26477.6

Arguments:

EARLYOUT: (Early Out flag)
NONE
FETCHMAX: (Override for FETCH MAXPAGES)
IGNORE
ISCANMAX: (Override for ISCAN MAXPAGES)
IGNORE

Predicates:

4) Predicate used in Join
Relational Operator: Equal (=)

Subquery Input Required: No
Filter Factor: 0.04

Predicate Text:

(Q10.S_NATIONKEY = Q8.N_NATIONKEY)

Input Streams:

19) From Operator #6

Estimated number of rows: 0.41902
Number of columns: 10
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0+Q9.PS_SUPPLYCOST+Q10.S_COMMENT
+Q10.S_PHONE+Q10.S_ADDRESS+Q10.S_NAME
+Q10.S_ACCTBAL+Q10.S_NATIONKEY+Q11.P_MFGR
+Q11.P_PARTKEY

21) From Operator #29

Estimated number of rows: 1
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q8.N_NAME

Output Streams:

22) To Operator #4

Estimated number of rows: 0.41902
Number of columns: 11
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0+Q8.N_NAME+Q9.PS_SUPPLYCOST
+Q10.S_COMMENT+Q10.S_PHONE+Q10.S_ADDRESS
+Q10.S_NAME+Q10.S_ACCTBAL+Q10.S_NATIONKEY
+Q11.P_MFGR+Q11.P_PARTKEY

6) NLJOIN: (Nested Loop Join)
Cumulative Total Cost: 713016
Cumulative CPU Cost: 8.00036e+09
Cumulative I/O Cost: 31800.2
Cumulative Re-Total Cost: 712741
Cumulative Re-CPU Cost: 8.00032e+09
Cumulative Re-I/O Cost: 31789.2
Cumulative First Row Cost: 713016
Estimated Bufferpool Buffers: 31790.2
Remote communication cost:26471.4

Arguments:

EARLYOUT: (Early Out flag)
NONE
FETCHMAX: (Override for FETCH MAXPAGES)
IGNORE
ISCANMAX: (Override for ISCAN MAXPAGES)
IGNORE
JN INPUT: (Join input leg)
OUTER

Predicates:

2) Predicate used in Join
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 1.16257e-05

Predicate Text:

(Q9.PS_SUPPLYCOST = Q6.\$C0)

Input Streams:

4) From Operator #7

Estimated number of rows: 36042.4
Number of columns: 9
Subquery predicate ID: Not Applicable

Column Names:

+Q9.PS_SUPPLYCOST+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q10.S_NAME+Q10.S_ACCTBAL
+Q10.S_NATIONKEY+Q11.P_MFGR+Q11.P_PARTKEY

18) From Operator #13

Estimated number of rows: 1.16257e-05

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0

Output Streams:

19) To Operator #5

Estimated number of rows: 0.41902

Number of columns: 10

Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0+Q9.PS_SUPPLYCOST+Q10.S_COMMENT

+Q10.S_PHONE+Q10.S_ADDRESS+Q10.S_NAME

+Q10.S_ACCTBAL+Q10.S_NATIONKEY+Q11.P_MFGR

+Q11.P_PARTKEY

7) SHIP : (Ship)

Cumulative Total Cost: 708746

Cumulative CPU Cost: 5.90826e+08

Cumulative I/O Cost: 31789.2

Cumulative Re-Total Cost: 708746

Cumulative Re-CPU Cost: 5.90826e+08

Cumulative Re-I/O Cost: 31789.2

Cumulative First Row Cost: 708746

Estimated Bufferpool Buffers: 31790.2

Remote communication cost:26449.2

Arguments:

CSERQY : (Remote common subexpression)

FALSE

DSTSEVER: (Destination (ship to) server)

- (NULL).

JN INPUT: (Join input leg)

OUTER

RMTQTX : (Remote statement)

SELECT A0."P_PARTKEY", A0."P_MFGR", A2."S_NATIONKEY", A2."S_ACCTBAL",
A2."S_NAME", A2."S_ADDRESS", A2."S_PHONE", A2."S_COMMENT", A1."PS_SUPPLYCOST"

```
FROM "TPCD"."PART" A0, "TPCD"."PARTSUPP" A1, "TPCD"."SUPPLIER" A2 WHERE
(A0."P_SIZE" = 15) AND (A0."P_TYPE" LIKE '%BRASS') AND (A0."P_PARTKEY" =
A1."PS_PARTKEY") AND (A2."S_SUPPKEY" = A1."PS_SUPPKEY") FOR READ ONLY
```

```
SRCSEVER: (Source (ship from) server)
```

```
DB2SERV
```

```
STREAM : (Remote stream)
```

```
FALSE
```

```
Input Streams:
```

```
-----
```

```
1) From Object DB2.PART
```

```
Estimated number of rows: 2e+06
```

```
Number of columns: 5
```

```
Subquery predicate ID: Not Applicable
```

```
Column Names:
```

```
-----
```

```
+Q11.$RID$+Q11.P_MFGR+Q11.P_TYPE+Q11.P_SIZE
```

```
+Q11.P_PARTKEY
```

```
2) From Object DB2.PARTSUPP
```

```
Estimated number of rows: 8e+06
```

```
Number of columns: 4
```

```
Subquery predicate ID: Not Applicable
```

```
Column Names:
```

```
-----
```

```
+Q9.$RID$+Q9.PS_SUPPLYCOST+Q9.PS_SUPPKEY
```

```
+Q9.PS_PARTKEY
```

```
3) From Object DB2.SUPPLIER
```

```
Estimated number of rows: 100000
```

```
Number of columns: 8
```

```
Subquery predicate ID: Not Applicable
```

```
Column Names:
```

```
-----
```

```
+Q10.$RID$+Q10.S_COMMENT+Q10.S_PHONE
```

```
+Q10.S_ADDRESS+Q10.S_NAME+Q10.S_ACCTBAL
```

```
+Q10.S_NATIONKEY+Q10.S_SUPPKEY
```

```
Output Streams:
```

```
-----
```

```
4) To Operator #6
```


Estimated number of rows: 36042.4
Number of columns: 9
Subquery predicate ID: Not Applicable

Column Names:

+Q9.PS_SUPPLYCOST+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q10.S_NAME+Q10.S_ACCTBAL
+Q10.S_NATIONKEY+Q11.P_MFGR+Q11.P_PARTKEY

13) FILTER: (Filter)

Cumulative Total Cost: 275.131
Cumulative CPU Cost: 242977
Cumulative I/O Cost: 11
Cumulative Re-Total Cost: 275.131
Cumulative Re-CPU Cost: 242977
Cumulative Re-I/O Cost: 11
Cumulative First Row Cost: 275.131
Estimated Bufferpool Buffers: 0
Remote communication cost:22.2188

Arguments:

JN INPUT: (Join input leg)
INNER

Predicates:

2) Residual Predicate
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 1.16257e-05

Predicate Text:

(Q9.PS_SUPPLYCOST = Q6.\$C0)

Input Streams:

17) From Operator #14

Estimated number of rows: 1
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0

Output Streams:

18) To Operator #6

Estimated number of rows: 1.16257e-05
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0

14) GRPBY : (Group By)

Cumulative Total Cost: 275.13
Cumulative CPU Cost: 241642
Cumulative I/O Cost: 11
Cumulative Re-Total Cost: 275.13
Cumulative Re-CPU Cost: 241642
Cumulative Re-I/O Cost: 11
Cumulative First Row Cost: 275.13
Estimated Bufferpool Buffers: 0
Remote communication cost:22.2188

Arguments:

AGGMODE : (Aggregation Mode)

COMPLETE

GROUPBYC: (Group By columns)

FALSE

GROUPBYN: (Number of Group By columns)

0

ONEFETCH: (One Fetch flag)

FALSE

Input Streams:

16) From Operator #15

Estimated number of rows: 0.8
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q5.PS_SUPPLYCOST

Output Streams:

17) To Operator #13

Estimated number of rows: 1

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0

15) MSJOIN: (Merge Scan Join)

Cumulative Total Cost: 275.13

Cumulative CPU Cost: 241192

Cumulative I/O Cost: 11

Cumulative Re-Total Cost: 275.13

Cumulative Re-CPU Cost: 241192

Cumulative Re-I/O Cost: 11

Cumulative First Row Cost: 275.13

Estimated Bufferpool Buffers: 0

Remote communication cost:22.2188

Arguments:

EARLYOUT: (Early Out flag)

NONE

INNERCOL: (Inner Order Columns)

1: Q2.N_NATIONKEY(A)

OUTERCOL: (Outer Order columns)

1: Q3.S_NATIONKEY(A)

TEMPSIZE: (Temporary Table Page Size)

4096

Predicates:

11) Predicate used in Join

Relational Operator: Equal (=)

Subquery Input Required: No

Filter Factor: 0.04

Predicate Text:

(Q3.S_NATIONKEY = Q2.N_NATIONKEY)

Input Streams:

9) From Operator #16

Estimated number of rows: 4
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q3.S_NATIONKEY(A)+Q4.PS_SUPPLYCOST

15) From Operator #22

Estimated number of rows: 0.2
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

Output Streams:

16) To Operator #14

Estimated number of rows: 0.8
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q5.PS_SUPPLYCOST

16) TBSCAN: (Table Scan)

Cumulative Total Cost: 275.103
Cumulative CPU Cost: 191030
Cumulative I/O Cost: 11
Cumulative Re-Total Cost: 25.0566
Cumulative Re-CPU Cost: 104891
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 275.101
Estimated Bufferpool Buffers: 0
Remote communication cost:10.8594

Arguments:

JN INPUT: (Join input leg)
OUTER
MAXPAGES: (Maximum pages for prefetch)
ALL
PREFETCH: (Type of Prefetch)
NONE
SCANDIR : (Scan Direction)
FORWARD

Input Streams:

8) From Operator #17

Estimated number of rows: 4
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q3.S_NATIONKEY(A)+Q4.PS_SUPPLYCOST

Output Streams:

9) To Operator #15

Estimated number of rows: 4
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q3.S_NATIONKEY(A)+Q4.PS_SUPPLYCOST

17) SORT : (Sort)

Cumulative Total Cost: 275.1
Cumulative CPU Cost: 185333
Cumulative I/O Cost: 11
Cumulative Re-Total Cost: 25.0535
Cumulative Re-CPU Cost: 99194
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 275.1
Estimated Bufferpool Buffers: 50012
Remote communication cost:10.8594

Arguments:

DUPLWARN: (Duplicates Warning flag)

```
FALSE
NUMROWS : (Estimated number of rows)
4
ROWWIDTH: (Estimated width of rows)
16
SORTKEY : (Sort Key column)
1: Q3.S_NATIONKEY(A)
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE  : (Uniqueness required flag)
FALSE
```

Input Streams:

7) From Operator #18

```
Estimated number of rows: 4
Number of columns: 2
Subquery predicate ID: Not Applicable
```

Column Names:

+Q3.S_NATIONKEY+Q4.PS_SUPPLYCOST

Output Streams:

8) To Operator #16

```
Estimated number of rows: 4
Number of columns: 2
Subquery predicate ID: Not Applicable
```

Column Names:

+Q3.S_NATIONKEY(A)+Q4.PS_SUPPLYCOST

18) SHIP : (Ship)

```
Cumulative Total Cost: 275.097
Cumulative CPU Cost: 180037
Cumulative I/O Cost: 11
Cumulative Re-Total Cost: 25.0535
Cumulative Re-CPU Cost: 99194
Cumulative Re-I/O Cost: 1
Cumulative First Row Cost: 125.05
Estimated Bufferpool Buffers: 50012
Remote communication cost:10.8594
```

```

Arguments:
-----
CSERQY  : (Remote common subexpression)
        FALSE
DSTSEVER: (Destination (ship to) server)
        - (NULL).
RMTQTXT : (Remote statement)
        SELECT A0."PS_SUPPLYCOST", A1."S_NATIONKEY" FROM "TPCD"."PARTSUPP"
A0, "TPCD"."SUPPLIER" A1 WHERE (:H0 = A0."PS_PARTKEY") AND (A1."S_SUPPKEY" =
A0."PS_SUPPKEY") FOR READ ONLY
SRCSEVER: (Source (ship from) server)
        DB2SERV
STREAM  : (Remote stream)
        FALSE

```

Input Streams:

5) From Object DB2.PARTSUPP

```

Estimated number of rows: 8e+06
Number of columns: 4
Subquery predicate ID: Not Applicable

```

Column Names:

```

-----
+Q4.$RID$+Q4.PS_SUPPLYCOST+Q4.PS_SUPPKEY
+Q4.PS_PARTKEY

```

6) From Object DB2.SUPPLIER

```

Estimated number of rows: 100000
Number of columns: 3
Subquery predicate ID: Not Applicable

```

Column Names:

```

-----
+Q3.$RID$+Q3.S_NATIONKEY+Q3.S_SUPPKEY

```

Output Streams:

7) To Operator #17

```

Estimated number of rows: 4
Number of columns: 2
Subquery predicate ID: Not Applicable

```

Column Names:

```

-----

```

+Q3.S_NATIONKEY+Q4.PS_SUPPLYCOST

22) FILTER: (Filter)

Cumulative Total Cost: 0.0247592
Cumulative CPU Cost: 45913.4
Cumulative I/O Cost: 0
Cumulative Re-Total Cost: 0.0103888
Cumulative Re-CPU Cost: 19265
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 0.022036
Estimated Bufferpool Buffers: 0
Remote communication cost:11.3594

Arguments:

JN INPUT: (Join input leg)
INNER

Predicates:

11) Residual Predicate

Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 0.04

Predicate Text:

(Q3.S_NATIONKEY = Q2.N_NATIONKEY)

Input Streams:

14) From Operator #23

Estimated number of rows: 5
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

Output Streams:

15) To Operator #15

Estimated number of rows: 0.2

Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

23) TBSCAN: (Table Scan)
Cumulative Total Cost: 0.0247592
Cumulative CPU Cost: 45913.4
Cumulative I/O Cost: 0
Cumulative Re-Total Cost: 0.0103888
Cumulative Re-CPU Cost: 19265
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 0.022036
Estimated Bufferpool Buffers: 0
Remote communication cost:11.3594

Arguments:

MAXPAGES: (Maximum pages for prefetch)
ALL
PREFETCH: (Type of Prefetch)
NONE
SCANDIR : (Scan Direction)
FORWARD

Input Streams:

13) From Operator #24

Estimated number of rows: 5
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

Output Streams:

14) To Operator #22

Estimated number of rows: 5
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

24) SORT : (Sort)
Cumulative Total Cost: 0.021261
Cumulative CPU Cost: 39426.4
Cumulative I/O Cost: 0
Cumulative Re-Total Cost: 0.00689066
Cumulative Re-CPU Cost: 12778
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 0.021261
Estimated Bufferpool Buffers: 2
Remote communication cost:11.3594

Arguments:

DUPLWARN: (Duplicates Warning flag)
FALSE
NUMROWS : (Estimated number of rows)
5
ROWWIDTH: (Estimated width of rows)
12
SORTKEY : (Sort Key column)
1: Q2.N_NATIONKEY(A)
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

12) From Operator #25

Estimated number of rows: 5
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY

Output Streams:

13) To Operator #23

Estimated number of rows: 5

Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

25) SHIP : (Ship)

Cumulative Total Cost: 0.0180496
Cumulative CPU Cost: 33471
Cumulative I/O Cost: 0
Cumulative Re-Total Cost: 0.00689066
Cumulative Re-CPU Cost: 12778
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 0.0103301
Estimated Bufferpool Buffers: 2
Remote communication cost:11.3594

Arguments:

CSERQY : (Remote common subexpression)
FALSE

DSTSEVER: (Destination (ship to) server)
- (NULL).

RMTQTX : (Remote statement)

SELECT A1."N_NATIONKEY" FROM "IITEST"."REGION" A0, "IITEST"."NATION"
A1 WHERE (A0."R_NAME" = 'EUROPE ') AND (A1."N_REGIONKEY" =
A0."R_REGIONKEY")

SRCSEVER: (Source (ship from) server)
ORASERV

STREAM : (Remote stream)
FALSE

Input Streams:

10) From Object ORA.REGION

Estimated number of rows: 5
Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:

+Q1.\$RID\$+Q1.R_NAME+Q1.R_REGIONKEY

11) From Object ORA.NATION

Estimated number of rows: 25

Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$+Q2.N_REGIONKEY+Q2.N_NATIONKEY

Output Streams:

12) To Operator #24

Estimated number of rows: 5
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY

29) SHIP : (Ship)

Cumulative Total Cost: 0.00680492
Cumulative CPU Cost: 12619
Cumulative I/O Cost: 0
Cumulative Re-Total Cost: 0.00448071
Cumulative Re-CPU Cost: 8309
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 0.0055959
Estimated Bufferpool Buffers: 1
Remote communication cost:9.35938

Arguments:

CSERQY : (Remote common subexpression)
FALSE
DSTSEVER: (Destination (ship to) server)
- (NULL).
JN INPUT: (Join input leg)
INNER
RMTQTX : (Remote statement)

**SELECT AO."N_NAME" FROM "IITEST"."NATION" AO WHERE (:HO =
AO."N_NATIONKEY")**

SRCSEVER: (Source (ship from) server)
ORASERV
STREAM : (Remote stream)
FALSE

Input Streams:

20) From Object ORA.NATION

Estimated number of rows: 25
Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:

+Q8.\$RID\$+Q8.N_NAME+Q8.N_NATIONKEY

Output Streams:

21) To Operator #5

Estimated number of rows: 1
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q8.N_NAME

31) SHIP : (Ship)

Cumulative Total Cost: 0.0047045
Cumulative CPU Cost: 8724
Cumulative I/O Cost: 0
Cumulative Re-Total Cost: 0.00240995
Cumulative Re-CPU Cost: 4469
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 0.00352514
Estimated Bufferpool Buffers: 1
Remote communication cost:9.35938

Arguments:

CSERQY : (Remote common subexpression)
FALSE
DSTSEVER: (Destination (ship to) server)
- (NULL).
JN INPUT: (Join input leg)
INNER
RMTQTX : (Remote statement)

SELECT '1' FROM "IITEST"."REGION" AO WHERE (AO."R_NAME" = 'EUROPE

')

SRCSEVER: (Source (ship from) server)
ORASERV

STREAM : (Remote stream)
FALSE

Input Streams:

25) From Object ORA.REGION

Estimated number of rows: 5
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q7.\$RID\$+Q7.R_NAME

Output Streams:

26) To Operator #2

Estimated number of rows: 1
Number of columns: 0
Subquery predicate ID: Not Applicable

Objects Used in Access Plan:

Schema: DB2I64
Name: ORDERSMQT
Type: Materialized View (reference only)

Schema: DB2
Name: PART
Type: Nickname
Time of creation: 2004-06-17-10.42.49.959240
Last statistics update:
Number of columns: 9
Number of rows: 2000000
Width of rows: 70
Number of buffer pool pages: 76238
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: DB2
Name: PARTSUPP
Type: Nickname
Time of creation: 2004-06-17-10.42.55.831083
Last statistics update:
Number of columns: 5
Number of rows: 8000000
Width of rows: 28
Number of buffer pool pages: 319290
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: DB2
Name: SUPPLIER
Type: Nickname
Time of creation: 2004-06-17-10.42.53.722466
Last statistics update:
Number of columns: 7
Number of rows: 100000
Width of rows: 157
Number of buffer pool pages: 4093
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: ORA
Name: NATION
Type: Nickname
Time of creation: 2004-06-16-23.22.43.109494
Last statistics update:
Number of columns: 4
Number of rows: 25
Width of rows: 62
Number of buffer pool pages: 15
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32

Container extent page count: 32

Schema: ORA
Name: REGION
Type: Nickname
Time of creation: 2004-06-16-23.22.43.342734
Last statistics update:
Number of columns: 3
Number of rows: 5
Width of rows: 56
Number of buffer pool pages: 15
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

The access plans generated in Example 4-50 on page 278 (DB2_MAXIMAL_PUSHDOWN = 'N') and Example 4-52 on page 310 (DB2_MAXIMAL_PUSHDOWN = 'Y') are different in that the 3-way join of PART, PARTSUPP, and SUPPLIER has now been pushed down to DB2 (see SHIP operator 7), thereby eliminating a need for a nested loop join. However, the estimated total cost with DB2_MAXIMAL_PUSHDOWN = 'Y' is now 713016 timerons, which is higher than the 696955 timerons with DB2_MAXIMAL_PUSHDOWN = 'N'.

Root cause of the problem

It appears that the problem query has some pushdownable predicates that were executed at the federated server, since the optimizer estimated it to be the lowest cost access plan for the query. This is because of the default setting of the DB2_MAXIMAL_PUSHDOWN server option.

Attention: In this scenario, the fact that the DB2 optimizer chose not to push down certain predicates is *not* indicative of a problem, and therefore *root cause* is a misnomer. However, given the lack of any other indications, this lack of pushdown of predicates merits investigation as a *possible* cause of the performance problem.

Apply best practices

We recommend that you consider the problem query with DB2_MAXIMAL_PUSHDOWN set to 'Y' using the SET SERVER OPTION to isolate the impact of this option only to the specific problem query.

Attention: In the final analysis, what really matters is not what the DB2 optimizer estimates to be the optimal access path based on timers, but the actual run times experienced by the user.

Since tuning tends to be a trial-and-error iterative process, it is more than likely that some of the options suggested could lead to other performance problems that would need to be investigated and resolved.

4.4.7 Default DB2_FENCED wrapper option with DPF

The DB2_FENCED wrapper option is new in DB2 II V8.2 and allows the DBA to decide whether the wrapper should operate in trusted or fenced mode. When the DB2_FENCED parameter is set to 'Y' and DB2 II is installed in a DPF environment, the federated server is able to generate access plans that parallelize the processing of nickname data, and thereby improve query performance. With the default setting of DB2_FENCED = 'N', there is no inter-partition parallelism for nickname data. For more details on the performance considerations of the DB2_FENCED wrapper option, refer to "Choice of DPF" on page 62.

In this scenario, we diagnose how the default setting of the DB2_FENCED wrapper option may inhibit a federated query accessing both local and nickname data in a DPF environment from delivering superior performance.

Triggering event

Users complained about poor response times with a specific query. The users claimed that they had never really experienced good performance from this query.

Hypotheses and validations

As before, since these were user complaints about the performance of a specific query, as per Figure 4-3 on page 122, we decided to enter the DB2 hypotheses hierarchy described in Figure 4-1 on page 117 and Example 4.2 on page 119, at a lower level, bypassing network- and system-related problems, and focused directly on federated application/query performance, as follows.

Hypothesis 1: Federated application/query performance

Before one can investigate the cause of a query's performance problem, one needs to identify the query in question. After identifying the query in question, one can begin diagnosing whether the performance problem is at the federated server, the remote data source, or equally divided between the two, as discussed in Example 4.2 on page 119.

- Identify the application and query.

In this case the user specifically identified the query in question as being the one shown in Example 4-53. It lists all orders by customer. The orders table is a local DB2 table, while the customer table is a nickname referencing a remote Oracle table.

Example 4-53 Problem query

```
select *
from tpcd.orders o, ora.customer c
where o.o_custkey = c.c_custkey
and c_mktsegment = 'AUTOMOBILE'
and c_nationkey > 20;
```

- Determine if the problem is at the Federated server or the data source.

Once the problem query has been identified, we need to determine if the query's performance problem is at the federated server, the remote data source, or distributed between both.

This information can be determined from a dynamic SQL snapshot, as discussed in 4.2.3, "Federated server or remote data source" on page 136. Example 4-54 is a dynamic SQL snapshot, and includes the problem query (highlighted) in the Statement text field.

Example 4-54 Dynamic SQL snapshot

get snapshot for dynamic sql on fedserv

Dynamic SQL Snapshot Result

Database name	= FEDSERV
Database path	= /data1/part/db2i64p/NODE0000/SQL00001/
Number of executions	= 1
Number of compilations	= 1
Worst preparation time (ms)	= 816
Best preparation time (ms)	= 816
Internal rows deleted	= 0
Internal rows inserted	= 0
Rows read	= 483185
Internal rows updated	= 0
Rows written	= 483185
Statement sorts	= 0
Statement sort overflows	= 0
Total sort time	= 0
Buffer pool data logical reads	= 4

```

Buffer pool data physical reads      = 1
Buffer pool temporary data logical reads  = 49862
Buffer pool temporary data physical reads = 0
Buffer pool index logical reads      = 6
Buffer pool index physical reads     = 3
Buffer pool temporary index logical reads = 0
Buffer pool temporary index physical reads = 0
Total execution time (sec.ms)      = 130.732961
Total user cpu time (sec.ms)      = 45.140000
Total system cpu time (sec.ms)    = 1.230000
Statement text                    = select * from tpcd.orders o, ora.customer
c where o.o_custkey = c.c_custkey and c_mktsegment = 'AUTOMOBILE' and
c_nationkey > 20

```

```

Number of executions                  = 1
Number of compilations                = 1
Worst preparation time (ms)           = 0
Best preparation time (ms)            = 0
Internal rows deleted                 = 0
Internal rows inserted                = 0
Rows read                          = 48238
Internal rows updated                 = 0
Rows written                          = 0
Statement sorts                      = 0
Buffer pool data logical reads        = 0
Buffer pool data physical reads       = 0
Buffer pool temporary data logical reads = 0
Buffer pool temporary data physical reads = 0
Buffer pool index logical reads       = 0
Buffer pool index physical reads      = 0
Buffer pool temporary index logical reads = 0
Buffer pool temporary index physical reads = 0
Total execution time (sec.ms)      = 6.373050
Total user cpu time (sec.ms)      = 0.000000
Total system cpu time (sec.ms)    = 0.000000
Statement text                    = SELECT AO."C_CUSTKEY", AO."C_NATIONKEY",
AO."C_NAME", AO."C_ADDRESS", AO."C_PHONE", AO."C_ACCTBAL", AO."C_COMMENT" FROM
"IITEST"."CUSTOMER" AO WHERE (AO."C_MKTSEGMENT" = 'AUTOMOBILE') AND (20 <
AO."C_NATIONKEY")

```

As discussed in 4.2.3, “Federated server or remote data source” on page 136, since there is no direct mechanism to link the remote SQL fragments in the dynamic cache with their corresponding user SQL statement, we generated **db2exfmt** output for the user query, as shown in Example 4-55, to obtain this link.

Example 4-55 db2exfmt output of problem query

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002

***** EXPLAIN INSTANCE *****

DB2_VERSION: 08.02.0
SOURCE_NAME: SQLC2E05
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-08-06-11.43.00.095214
EXPLAIN_REQUESTER: DB2I64P

Database Context:

Parallelism: Inter-Partition Parallelism

CPU Speed: 5.313873e-07
Comm Speed: 100
Buffer Pool size: 75000
Sort Heap size: 256
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 640

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 203 -----

QUERYNO: 1
QUERYTAG:
Statement Type: Select
Updatable: No
Deletable: No
Query Degree: 1

Original Statement:

select *
from tpcd.orders o, ora.customer c
where o.o_custkey = c.c_custkey and c_mktsegment = 'AUTOMOBILE' and

c_nationkey > 20

Optimized Statement:

```
-----  
SELECT Q3.$C7 AS "O_ORDERKEY", Q3.$C8 AS "O_CUSTKEY", Q3.$C6 AS  
      "O_ORDERSTATUS", Q3.$C5 AS "O_TOTALPRICE", Q3.$C4 AS "O_ORDERDATE",  
      Q3.$C3 AS "O_ORDERPRIORITY", Q3.$C2 AS "O_CLERK", Q3.$C1 AS  
      "O_SHIPPRIORITY", Q3.$C0 AS "O_COMMENT", Q1.C_CUSTKEY AS "C_CUSTKEY",  
      Q1.C_NAME AS "C_NAME", Q1.C_ADDRESS AS "C_ADDRESS", Q1.C_NATIONKEY AS  
      "C_NATIONKEY", Q1.C_PHONE AS "C_PHONE", Q1.C_ACCTBAL AS "C_ACCTBAL",  
      'AUTOMOBILE' AS "C_MKTSEGMENT", Q1.C_COMMENT AS "C_COMMENT"  
FROM ORA.CUSTOMER AS Q1,  
      (SELECT Q2.O_COMMENT, Q2.O_SHIPPRIORITY, Q2.O_CLERK, Q2.O_ORDERPRIORITY,  
              Q2.O_ORDERDATE, Q2.O_TOTALPRICE, Q2.O_ORDERSTATUS, Q2.O_ORDERKEY,  
              Q2.O_CUSTKEY  
      FROM TPCD.ORDERS AS Q2) AS Q3  
WHERE (20 < Q1.C_NATIONKEY) AND (Q1.C_MKTSEGMENT = 'AUTOMOBILE') AND (Q3.$C8  
      = Q1.C_CUSTKEY)
```

Access Plan:

Total Cost: 2.29808e+06

Query Degree:1

```
      Rows  
      RETURN  
      ( 1)  
      Cost  
      I/O  
      |  
      2.18739e+06  
      HSJOIN  
      ( 2)  
      2.29808e+06  
      572713  
      /-----+-----\  
3.00114e+07      144000  
DTQ              SHIP  
( 3)            ( 5)  
638128          128168  
296184          32173  
|              |  
1.00038e+07      1.5e+06  
TBSCAN          NICKNM: ORA  
( 4)            CUSTOMER  
628410  
296184  
|
```

1.00038e+07
TABLE: TPCD
ORDERS

1) RETURN: (Return Result)
Cumulative Total Cost: 2.29808e+06
Cumulative CPU Cost: 5.3062e+10
Cumulative I/O Cost: 572713
Cumulative Re-Total Cost: 2.29808e+06
Cumulative Re-CPU Cost: 5.3062e+10
Cumulative Re-I/O Cost: 572713
Cumulative First Row Cost: 2.29808e+06
Cumulative Comm Cost: 2.41827e+06
Cumulative First Comm Cost: 0
Estimated Bufferpool Buffers: 474290
Remote communication cost: 113565

Arguments:

BLDLEVEL: (Build level)
DB2 v8.1.1.64 : s040509
STMTHCAP: (Statement heap size)
4096

Input Streams:

6) From Operator #2

Estimated number of rows: 2.18739e+06
Partition Map ID: -100
Partitioning: (COOR)
Coordinator Partition
Number of columns: 17
Subquery predicate ID: Not Applicable

Column Names:

+Q4.C_COMMENT+Q4.C_MKTSEGMENT+Q4.C_ACCTBAL
+Q4.C_PHONE+Q4.C_NATIONKEY+Q4.C_ADDRESS
+Q4.C_NAME+Q4.C_CUSTKEY+Q4.O_COMMENT
+Q4.O_SHIPPRIORITY+Q4.O_CLERK
+Q4.O_ORDERPRIORITY+Q4.O_ORDERDATE
+Q4.O_TOTALPRICE+Q4.O_ORDERSTATUS+Q4.O_CUSTKEY
+Q4.O_ORDERKEY

Partition Column Names:

+NONE

2) HSJOIN: (Hash Join)
Cumulative Total Cost: 2.29808e+06
Cumulative CPU Cost: 5.3062e+10
Cumulative I/O Cost: 572713
Cumulative Re-Total Cost: 2.29808e+06
Cumulative Re-CPU Cost: 5.3062e+10
Cumulative Re-I/O Cost: 572713
Cumulative First Row Cost: 2.29808e+06
Cumulative Comm Cost: 2.41827e+06
Cumulative First Comm Cost: 0
Estimated Bufferpool Buffers: 474290
Remote communication cost: 113565

Arguments:

BITFLTR : (Hash Join Bit Filter used)
142240
EARLYOUT: (Early Out flag)
LEFT
HASHCODE: (Hash Code Size)
24 BIT
TEMPSIZE: (Temporary Table Page Size)
4096

Predicates:

4) Predicate used in Join
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 5.06148e-07

Predicate Text:

(Q3.\$C8 = Q1.C_CUSTKEY)

Input Streams:

3) From Operator #3

Estimated number of rows: 3.00114e+07
Partition Map ID: -100
Partitioning: (COOR)
Coordinator Partition
Number of columns: 9

Subquery predicate ID: Not Applicable

Column Names:

+Q3.O_COMMENT+Q3.O_SHIPPRIORITY+Q3.O_CLERK
+Q3.O_ORDERPRIORITY+Q3.O_ORDERDATE
+Q3.O_TOTALPRICE+Q3.O_ORDERSTATUS
+Q3.O_ORDERKEY+Q3.O_CUSTKEY

Partition Column Names:

+NONE

5) From Operator #5

Estimated number of rows: 144000

Partition Map ID: -100

Partitioning: (COOR)

Coordinator Partition

Number of columns: 7

Subquery predicate ID: Not Applicable

Column Names:

+Q1.C_COMMENT+Q1.C_ACCTBAL+Q1.C_PHONE
+Q1.C_ADDRESS+Q1.C_NAME+Q1.C_NATIONKEY
+Q1.C_CUSTKEY

Partition Column Names:

+NONE

Output Streams:

6) To Operator #1

Estimated number of rows: 2.18739e+06

Partition Map ID: -100

Partitioning: (COOR)

Coordinator Partition

Number of columns: 17

Subquery predicate ID: Not Applicable

Column Names:

+Q4.C_COMMENT+Q4.C_MKTSEGMENT+Q4.C_ACCTBAL
+Q4.C_PHONE+Q4.C_NATIONKEY+Q4.C_ADDRESS
+Q4.C_NAME+Q4.C_CUSTKEY+Q4.O_COMMENT


```
+Q4.0_SHIPPRIORITY+Q4.0_CLERK
+Q4.0_ORDERPRIORITY+Q4.0_ORDERDATE
+Q4.0_TOTALPRICE+Q4.0_ORDERSTATUS+Q4.0_CUSTKEY
+Q4.0_ORDERKEY
```

Partition Column Names:

+NONE

3) TQ : (Table Queue)

```
Cumulative Total Cost: 638128
Cumulative CPU Cost: 3.73425e+10
Cumulative I/O Cost: 296184
Cumulative Re-Total Cost: 628410
Cumulative Re-CPU Cost: 1.90555e+10
Cumulative Re-I/O Cost: 296184
Cumulative First Row Cost: 10.4828
Cumulative Comm Cost:2.41827e+06
Cumulative First Comm Cost:0
Estimated Bufferpool Buffers: 296184
```

Arguments:

JN INPUT: (Join input leg)
OUTER
LISTENER: (Listener Table Queue type)
FALSE
TQMERGE : (Merging Table Queue flag)
FALSE
TQREAD : (Table Queue Read type)
READ AHEAD
TQSEND : (Table Queue Write type)
DIRECTED
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

2) From Operator #4

```
Estimated number of rows: 1.00038e+07
Partition Map ID: 4
Partitioning: (MULT )
Multiple Partitions
Number of columns: 9
Subquery predicate ID: Not Applicable
```

Column Names:

```

-----
+Q3.0_COMMENT+Q3.0_SHIPPRIORITY+Q3.0_CLERK
+Q3.0_ORDERPRIORITY+Q3.0_ORDERDATE
+Q3.0_TOTALPRICE+Q3.0_ORDERSTATUS
+Q3.0_ORDERKEY+Q3.0_CUSTKEY

```

Partition Column Names:

```

-----
+1: Q3.0_ORDERKEY

```

Output Streams:

3) To Operator #2

```

Estimated number of rows: 3.00114e+07
Partition Map ID: -100
Partitioning: (COOR )
Coordinator Partition
Number of columns: 9
Subquery predicate ID: Not Applicable

```

Column Names:

```

-----
+Q3.0_COMMENT+Q3.0_SHIPPRIORITY+Q3.0_CLERK
+Q3.0_ORDERPRIORITY+Q3.0_ORDERDATE
+Q3.0_TOTALPRICE+Q3.0_ORDERSTATUS
+Q3.0_ORDERKEY+Q3.0_CUSTKEY

```

Partition Column Names:

```

-----
+NONE

```

4) TBSCAN: (Table Scan)

```

Cumulative Total Cost: 628410
Cumulative CPU Cost: 1.90555e+10
Cumulative I/O Cost: 296184
Cumulative Re-Total Cost: 628410
Cumulative Re-CPU Cost: 1.90555e+10
Cumulative Re-I/O Cost: 296184
Cumulative First Row Cost: 10.4278
Cumulative Comm Cost:0
Cumulative First Comm Cost:0
Estimated Bufferpool Buffers: 296184

```

Arguments:

```

-----
MAXPAGES: (Maximum pages for prefetch)

```

ALL
PREFETCH: (Type of Prefetch)
 SEQUENTIAL
ROWLOCK : (Row Lock intent)
 NEXT KEY SHARE
SCANDIR : (Scan Direction)
 FORWARD
TABLOCK : (Table Lock intent)
 INTENT SHARE
TBISOLVL: (Table access Isolation Level)
 CURSOR STABILITY

Input Streams:

1) From Object TPCD.ORDERS

Estimated number of rows: 1.00038e+07

Partition Map ID: 4

Partitioning: (MULT)

Multiple Partitions

Number of columns: 10

Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$+Q2.0_CUSTKEY+Q2.0_ORDERKEY

+Q2.0_ORDERSTATUS+Q2.0_TOTALPRICE

+Q2.0_ORDERDATE+Q2.0_ORDERPRIORITY+Q2.0_CLERK

+Q2.0_SHIPPRIORITY+Q2.0_COMMENT

Partition Column Names:

+1: Q2.0_ORDERKEY

Output Streams:

2) To Operator #3

Estimated number of rows: 1.00038e+07

Partition Map ID: 4

Partitioning: (MULT)

Multiple Partitions

Number of columns: 9

Subquery predicate ID: Not Applicable

Column Names:

+Q3.0_COMMENT+Q3.0_SHIPPRIORITY+Q3.0_CLERK

```
+Q3.0_ORDERPRIORITY+Q3.0_ORDERDATE
+Q3.0_TOTALPRICE+Q3.0_ORDERSTATUS
+Q3.0_ORDERKEY+Q3.0_CUSTKEY
```

Partition Column Names:

```
-----
+1: Q3.0_ORDERKEY
```

5) SHIP : (Ship)

```
Cumulative Total Cost: 128168
Cumulative CPU Cost: 4.29461e+09
Cumulative I/O Cost: 32173
Cumulative Re-Total Cost: 2176.51
Cumulative Re-CPU Cost: 4.0959e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 25.8328
Cumulative Comm Cost:0
Cumulative First Comm Cost:0
Estimated Bufferpool Buffers: 32173
Remote communication cost:113565
```

Arguments:

```
-----
CSERQY : (Remote common subexpression)
FALSE
DSTSEVER: (Destination (ship to) server)
- (NULL).
JN INPUT: (Join input leg)
INNER
```

RMTQTX : (Remote statement)

```
SELECT AO."C_CUSTKEY", AO."C_NATIONKEY", AO."C_NAME", AO."C_ADDRESS",
AO."C_PHONE", AO."C_ACCTBAL", AO."C_COMMENT" FROM "IITEST"."CUSTOMER" AO WHERE
(AO."C_MKTSEGMENT" = 'AUTOMOBILE') AND (20 < AO."C_NATIONKEY")
```

SRCSEVER: (Source (ship from) server)

ORASERV

STREAM : (Remote stream)

FALSE

Input Streams:

```
-----
4) From Object ORA.CUSTOMER
```

Estimated number of rows: 1.5e+06

Partition Map ID: -100

Partitioning: (COOR)

Coordinator Partition

Number of columns: 9

Subquery predicate ID: Not Applicable

Column Names:

+Q1.\$RID\$+Q1.C_COMMENT+Q1.C_ACCTBAL+Q1.C_PHONE
+Q1.C_ADDRESS+Q1.C_NAME+Q1.C_NATIONKEY
+Q1.C_MKTSEGMENT+Q1.C_CUSTKEY

Partition Column Names:

+NONE

Output Streams:

5) To Operator #2

Estimated number of rows: 144000
Partition Map ID: -100
Partitioning: (COOR)
Coordinator Partition
Number of columns: 7
Subquery predicate ID: Not Applicable

Column Names:

+Q1.C_COMMENT+Q1.C_ACCTBAL+Q1.C_PHONE
+Q1.C_ADDRESS+Q1.C_NAME+Q1.C_NATIONKEY
+Q1.C_CUSTKEY

Partition Column Names:

+NONE

Objects Used in Access Plan:

Schema: ORA

Name: CUSTOMER

Type: Nickname

Time of creation: 2004-06-10-08.32.16.950465

Last statistics update:

Number of columns: 8

Number of rows: 1500000

Width of rows: 236

Number of buffer pool pages: 32173

Distinct row values: No

Tablespace name:

Tablespace overhead: 24.100000

Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: TPCD

Name: ORDERS

Type: Table

Time of creation: 2004-06-04-07.02.15.736633
Last statistics update: 2004-06-09-23.20.10.138687
Number of columns: 9
Number of rows: 10003789
Width of rows: 115
Number of buffer pool pages: 296184
Distinct row values: No
Tablespace name: DATA_TS
Tablespace overhead: 9.500000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 48
Container extent page count: 16
Table overflow record count: 0
Table Active Blocks: -1

Note: The DB2 instance used is DB2I64P and the Parallelism field in the Database Context section indicates that inter-partition parallelism is enabled, indicating a DPF environment.

The RMTQTX fields of SHIP operator 5 in Example 4-55 on page 341 contain the following remote SQL fragment text:

```
SELECT AO."C_CUSTKEY", AO."C_NATIONKEY", AO."C_NAME", AO."C_ADDRESS",  
AO."C_PHONE", AO."C_ACCTBAL", AO."C_COMMENT" FROM "IITEST"."CUSTOMER" AO  
WHERE (AO."C_MKTSEGMENT" = 'AUTOMOBILE') AND (20 < AO."C_NATIONKEY")
```

This remote SQL fragment is located in the dynamic cache output shown in Example 4-54 on page 340, and the following information can be gathered:

- **Number of executions** is 1 for the user-entered query as well as the single remote SQL fragment.
- **Total execution time (sec.ms)**, which is 130.732961 seconds for the user-entered query, and 6.373050 seconds for the remote fragment.
- **Total user cpu time (sec.ms)** and **Total system cpu time (sec.ms)**, which are 45.140000 and 1.230000 for the user-entered query, and zero for the remote SQL fragment.

- **Rows read** is 483185 for the user-entered query, which is the number of rows returned to the user; and 48238 rows for the remote SQL fragment, which indicates the number of rows returned to the federated server from the remote data source.
- Other fields of interest include **Statement sorts**, **Statement sort overflows**, and **Total sort time**, which only apply to the user-entered query, and all have zero values.
- The buffer pool hit ratio appears to be 100 percent.

Note: To obtain the average elapsed and CPU times, as well as the number of rows returned, you must divide the values shown by the **Number of executions**.

We can derive the following information from these metrics:

- The average number of rows returned from the remote data source to the federated server is $(48238 / 1) = 48238$.
- The average elapsed time for the user query is $(130.732961 / 1) = 130.732961$ seconds, while that of the remote SQL fragment is $(6.373050 / 1) = 6.373050$ seconds.

Attention: In our example, given that the total query elapsed time is 130.732961 seconds and the time spent at the remote data source is only 6.373050 seconds, it is clear that the predominant portion of the query time is at the federated server. The time spent at the federated server is $(130.732961 - 6.373050) = 124.359911$ seconds.

This performance problem, therefore, needs to be investigated at the federated server only, as described in 4.2.4, “Federated server related” on page 152.

► **Federated server related.**

As discussed in 4.2.4, “Federated server related” on page 152, poor performance at the federated server may be caused by a number of factors such as statistics, index information, pushdown, joins, and parallelism.

Each of these factors needs to be evaluated in turn to determine the root cause of the performance problem.

- A review of the number of rows (144000) estimated by the DB2 optimizer in Example 4-55 on page 341 as being returned from the remote data source to the federated server, and the actual number of rows returned (**Rows read** field 48238) from the remote data source in Example 4-54 on page 340 shows a discrepancy that merits investigation. Since this

process has already been discussed in 4.4.2, “Missing or incorrect statistics/index information” on page 170, we will not be repeating it in this scenario.

Note: Table TPCD.ORDERS is a local table with an estimated 1.00038e+07 rows, and the dynamic SQL snapshot does not provide information about the actual number of rows returned.

- A review of the remote SQL fragment text and the original statement (which is a join of a nickname and local DPF data, as shown in Example 4-55 on page 341) shows that there are no predicates that need to be investigated from a pushdown perspective.

The join predicate has to be executed at the federated server because the join is of a local table and a nickname referencing a remote data source.

- **db2exfmt** output in Example 4-55 on page 341 shows the hash join (HSJOIN operator 2) being performed with an estimated 3.00114e+07 rows from the TPCD.ORDERS outer table with an estimated 144000 rows from the inner table ORA.CUSTOMER. The local TPCD.ORDERS table is scanned (TBSCAN operator 4) in parallel (DTQ operator 3) and data is sent to the coordinator partition where the hash join is performed.
- As mentioned earlier, **db2exfmt** output in Example 4-55 on page 341 shows that inter-partition parallelism is enabled, but that nickname data ORA.CUSTOMER is serially joined with local TPCD.ORDERS data at the coordinator partition.

If the hash join (HSJOIN operator 2) operation were to be done in the partitions where the TPCD.ORDERS data is located rather than at the coordinator node, a TQ operator would appear above the HSJOIN operator in the access plan graph. If the parallel plan that would do the join in the partitions is allowed, it is still the optimizer's choice based on cost to decide on whether to use that plan or a non-parallel plan such as the one in the example. We therefore need to determine whether inter-partition parallelism has been inhibited by the DB2_FENCED wrapper option or the decision not to perform joins in parallel was a cost optimization decision by the DB2 optimizer. Example 4-56 shows the query for determining the NET8 wrapper's DB2_FENCED option setting. The result shows a default setting of 'N' (trusted), which inhibits inter-partition parallelism with nickname data.

Note: **db2exfmt** output in Example 4-55 on page 341 shows Query Degree:1 indicating that intra-partition parallelism is disabled.

The Computation Partition Group (CPG) capability does not apply in this case because the problem query joins nickname and local DPF data, and CPG applies only when both joined tables are nicknames.

Example 4-56 Show wrapper option

```
select substr(wrapname,1,8) as wrapname,  
substr(option, 1,12) as option,  
substr(setting, 1,8) as setting  
from syscat.wrapoptions  
where wrapname = 'NET8'
```

WRAPNAME	OPTION	SETTING
NET8	DB2_FENCED	N

1 record(s) selected.

Root cause of the problem

It appears that the problem query *cannot* exploit inter-partition parallel processing with nickname data because the Oracle wrapper (NET8) has its wrapper option DB2_FENCED default to 'N' (trusted) instead of fenced (DB2_FENCED = 'Y').

Apply best practices

We recommend the following steps to address the performance impact of the default DB2_FENCED wrapper option on the problem query.

1. Change the Oracle wrapper (NET8) DB2_FENCED option from its default value of 'N' (trusted) to 'Y' (fenced).

Example 4-57 shows how the DB2_FENCED wrapper option can be altered to 'Y' (fenced).

Example 4-57 Alter the wrapper option DB2_FENCED to 'Y'

```
connect to fedserv
```

Database Connection Information

```
Database server          = DB2/AIX64 8.2.0  
SQL authorization ID     = DB2I64P  
Local database alias     = FEDSERV
```

```
alter wrapper net8 options (set DB2_FENCED 'Y')  
DB20000I The SQL command completed successfully.
```

Note: We chose not to change the database configuration parameter DFT_DEGREE setting to -1 or ANY, or the database manager configuration parameter INTRA_PARALLEL to YES to enable intra-partition parallelism, since the emphasis of this scenario is on the performance impact of the default DB2_FENCED wrapper option.

2. Run **db2exfmt** on the query.

Example 4-58 on page 356 shows the **db2exfmt** output after the wrapper option was changed to fenced from trusted. The Access Plan section shows a directed table queue operation (DTQ operator 2) above the nested loop join (NLJOIN operator 3) that indicates that the federated server distributes (as indicated by the broadcast table queue BTQ operator 4) nickname data from ORA.CUSTOMER to the other partitions and performs the join with the local data in parallel. The Total Cost is estimated to be 136545 timerons, which is considerably less than the access plan estimate of 2.29808e+06 timerons in Example 4-55 on page 341.

Note: The choice of the parallel plan is made by the optimizer based on cost. We can see the parallel plan that was selected after we set the wrapper option DB2_FENCED to 'Y', and the non-parallel plan that was selected when the DB2_FENCED was set to 'N'. We can also see the estimated cumulative cost (timerons) of both plans. The cost of the non-parallel plan is 2.29808e+06 (that is, 2,298,080) timerons, while that of the parallel plan is 136,545 timerons. In this case the parallel plan is lower in cost, which is the reason for its selection for this query once we set DB2_FENCED = 'Y'. If the parallel plan had not been the lower-cost plan, we would still have seen the non-parallel plan in **db2exfmt** after setting DB2_FENCED 'Y'. The parallel plan would still have been evaluated by the optimizer, but would not have been selected due to its higher-cost estimate, and would therefore not appear in **db2exfmt**.

Attention: In the final analysis, what really matters is not what the DB2 optimizer estimates to be the optimal access path based on timerons, but the actual run times experienced by the user.

Since tuning tends to be a trial-and-error iterative process, it is more than likely that some of the options suggested could lead to other performance problems that would need to be investigated and resolved.

Example 4-58 db2exfmt output of problem query

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002

***** EXPLAIN INSTANCE *****

DB2_VERSION: 08.02.0
SOURCE_NAME: SQLC2E05
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-08-06-11.45.14.176011
EXPLAIN_REQUESTER: DB2I64P

Database Context:

Parallelism: Inter-Partition Parallelism

CPU Speed: 5.313873e-07
Comm Speed: 100
Buffer Pool size: 75000
Sort Heap size: 256
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 640

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 203 -----

QUERYNO: 1
QUERYTAG:
Statement Type: Select
Updatable: No
Deletable: No
Query Degree: 1

Original Statement:

select *
from tpcd.orders o, ora.customer c
where o.o_custkey = c.c_custkey and c_mktsegment = 'AUTOMOBILE' and

c_nationkey > 20

Optimized Statement:

```
-----
SELECT Q2.O_ORDERKEY AS "O_ORDERKEY", Q2.O_CUSTKEY AS "O_CUSTKEY",
       Q2.O_ORDERSTATUS AS "O_ORDERSTATUS", Q2.O_TOTALPRICE AS
       "O_TOTALPRICE", Q2.O_ORDERDATE AS "O_ORDERDATE", Q2.O_ORDERPRIORITY
       AS "O_ORDERPRIORITY", Q2.O_CLERK AS "O_CLERK", Q2.O_SHIPPRIORITY AS
       "O_SHIPPRIORITY", Q2.O_COMMENT AS "O_COMMENT", Q1.C_CUSTKEY AS
       "C_CUSTKEY", Q1.C_NAME AS "C_NAME", Q1.C_ADDRESS AS "C_ADDRESS",
       Q1.C_NATIONKEY AS "C_NATIONKEY", Q1.C_PHONE AS "C_PHONE",
       Q1.C_ACCTBAL AS "C_ACCTBAL", 'AUTOMOBILE' AS "C_MKTSEGMENT",
       Q1.C_COMMENT AS "C_COMMENT"
FROM ORA.CUSTOMER AS Q1, TPCD.ORDERS AS Q2
WHERE (20 < Q1.C_NATIONKEY) AND (Q1.C_MKTSEGMENT = 'AUTOMOBILE') AND
      (Q2.O_CUSTKEY = Q1.C_CUSTKEY)
```

Access Plan:

Total Cost: 136545
Query Degree:1

Rows	
RETURN	
(1)	
Cost	
I/O	
2.18739e+06	
DTQ	
(2)	
136545	
32193.1	
729129	
NLJOIN	
(3)	
135482	
32193.1	
/-----+-----\	
144000	5.0634
BTQ	FETCH
(4)	(7)
128290	53.8812
32173	17.5276
	/---+---\
144000	5.0634 1.00038e+07
SHIP	RIDSCN TABLE: TPCD

(5)	(8)	ORDERS
128168	20.8488	
32173	2	
1.5e+06	5.0634	
NICKNM: ORA	SORT	
CUSTOMER	(9)	
	20.8481	
	2	
	5.0634	
	IXSCAN	
	(10)	
	20.8465	
	2	
	1.00038e+07	
	INDEX: TPCD	
	O_CK	

1) RETURN: (Return Result)

Cumulative Total Cost: 136545

Cumulative CPU Cost: 1.99431e+10

Cumulative I/O Cost: 32193.1

Cumulative Re-Total Cost: 9306.44

Cumulative Re-CPU Cost: 1.75135e+10

Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 57.1984

Cumulative Comm Cost: 589375

Cumulative First Comm Cost: 0

Estimated Bufferpool Buffers: 32191.1

Remote communication cost: 113565

Arguments:

BLDLEVEL: (Build level)

DB2 v8.1.1.64 : s040509

STMHEAP: (Statement heap size)

4096

Input Streams:

11) From Operator #2

Estimated number of rows: 2.18739e+06

Partition Map ID: -100

Partitioning: (COOR)
Coordinator Partition
Number of columns: 17
Subquery predicate ID: Not Applicable

Column Names:

+Q3.C_COMMENT+Q3.C_MKTSEGMENT+Q3.C_ACCTBAL
+Q3.C_PHONE+Q3.C_NATIONKEY+Q3.C_ADDRESS
+Q3.C_NAME+Q3.C_CUSTKEY+Q3.O_COMMENT
+Q3.O_SHIPPRIORITY+Q3.O_CLERK
+Q3.O_ORDERPRIORITY+Q3.O_ORDERDATE
+Q3.O_TOTALPRICE+Q3.O_ORDERSTATUS+Q3.O_CUSTKEY
+Q3.O_ORDERKEY

Partition Column Names:

+NONE

2) TQ : (Table Queue)

Cumulative Total Cost: 136545
Cumulative CPU Cost: 1.99431e+10
Cumulative I/O Cost: 32193.1
Cumulative Re-Total Cost: 9306.44
Cumulative Re-CPU Cost: 1.75135e+10
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 57.1984
Cumulative Comm Cost: 589375
Cumulative First Comm Cost: 0
Estimated Bufferpool Buffers: 32191.1
Remote communication cost: 113565

Arguments:

LISTENER: (Listener Table Queue type)
FALSE
TQMERGE : (Merging Table Queue flag)
FALSE
TQREAD : (Table Queue Read type)
READ AHEAD
TQSEND : (Table Queue Write type)
DIRECTED
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

10) From Operator #3

Estimated number of rows: 729129
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 17
Subquery predicate ID: Not Applicable

Column Names:

+Q3.C_COMMENT+Q3.C_MKTSEGMENT+Q3.C_ACCTBAL
+Q3.C_PHONE+Q3.C_NATIONKEY+Q3.C_ADDRESS
+Q3.C_NAME+Q3.C_CUSTKEY+Q3.O_COMMENT
+Q3.O_SHIPRIORITY+Q3.O_CLERK
+Q3.O_ORDERPRIORITY+Q3.O_ORDERDATE
+Q3.O_TOTALPRICE+Q3.O_ORDERSTATUS+Q3.O_CUSTKEY
+Q3.O_ORDERKEY

Partition Column Names:

+1: Q3.O_ORDERKEY

Output Streams:

11) To Operator #1

Estimated number of rows: 2.18739e+06
Partition Map ID: -100
Partitioning: (COOR)
Coordinator Partition
Number of columns: 17
Subquery predicate ID: Not Applicable

Column Names:

+Q3.C_COMMENT+Q3.C_MKTSEGMENT+Q3.C_ACCTBAL
+Q3.C_PHONE+Q3.C_NATIONKEY+Q3.C_ADDRESS
+Q3.C_NAME+Q3.C_CUSTKEY+Q3.O_COMMENT
+Q3.O_SHIPRIORITY+Q3.O_CLERK
+Q3.O_ORDERPRIORITY+Q3.O_ORDERDATE
+Q3.O_TOTALPRICE+Q3.O_ORDERSTATUS+Q3.O_CUSTKEY
+Q3.O_ORDERKEY

Partition Column Names:

+NONE

```

3) NLJOIN: (Nested Loop Join)
Cumulative Total Cost: 135482
Cumulative CPU Cost: 1.79422e+10
Cumulative I/O Cost: 32193.1
Cumulative Re-Total Cost: 9306.44
Cumulative Re-CPU Cost: 1.75135e+10
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 57.1435
Cumulative Comm Cost:68135.2
Cumulative First Comm Cost:0
Estimated Bufferpool Buffers: 32191.1
Remote communication cost:113565

```

Arguments:

```

-----
EARLYOUT: (Early Out flag)
    NONE
FETCHMAX: (Override for FETCH MAXPAGES)
    IGNORE
ISCANMAX: (Override for ISCAN MAXPAGES)
    IGNORE

```

Predicates:

```

-----
4) Predicate used in Join
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 5.06148e-07

```

Predicate Text:

```

-----
(Q2.0_CUSTKEY = Q1.C_CUSTKEY)

```

Input Streams:

```

-----
3) From Operator #4

Estimated number of rows: 144000
Partition Map ID: 4
Partitioning: (MULT )
Multiple Partitions
Number of columns: 7
Subquery predicate ID: Not Applicable

Column Names:
-----
+Q1.C_COMMENT+Q1.C_ACCTBAL+Q1.C_PHONE
+Q1.C_ADDRESS+Q1.C_NAME+Q1.C_NATIONKEY

```


+Q1.C_CUSTKEY

Partition Column Names:

+NONE

9) From Operator #7

Estimated number of rows: 5.0634

Partition Map ID: 4

Partitioning: (MULT)

Multiple Partitions

Number of columns: 10

Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$+Q2.0_COMMENT+Q2.0_SHIPPRIORITY

+Q2.0_CLERK+Q2.0_ORDERPRIORITY+Q2.0_ORDERDATE

+Q2.0_TOTALPRICE+Q2.0_ORDERSTATUS

+Q2.0_ORDERKEY+Q2.0_CUSTKEY

Partition Column Names:

+1: Q2.0_ORDERKEY

Output Streams:

10) To Operator #2

Estimated number of rows: 729129

Partition Map ID: 4

Partitioning: (MULT)

Multiple Partitions

Number of columns: 17

Subquery predicate ID: Not Applicable

Column Names:

+Q3.C_COMMENT+Q3.C_MKTSEGMENT+Q3.C_ACCTBAL

+Q3.C_PHONE+Q3.C_NATIONKEY+Q3.C_ADDRESS

+Q3.C_NAME+Q3.C_CUSTKEY+Q3.0_COMMENT

+Q3.0_SHIPPRIORITY+Q3.0_CLERK

+Q3.0_ORDERPRIORITY+Q3.0_ORDERDATE

+Q3.0_TOTALPRICE+Q3.0_ORDERSTATUS+Q3.0_CUSTKEY

+Q3.0_ORDERKEY

Partition Column Names:

+1: Q3.0_ORDERKEY

4) TQ : (Table Queue)

Cumulative Total Cost: 128290
Cumulative CPU Cost: 4.52455e+09
Cumulative I/O Cost: 32173
Cumulative Re-Total Cost: 2176.51
Cumulative Re-CPU Cost: 4.0959e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 25.8878
Cumulative Comm Cost: 68135.2
Cumulative First Comm Cost: 0
Estimated Bufferpool Buffers: 32173
Remote communication cost: 113565

Arguments:

JN INPUT: (Join input leg)
OUTER
LISTENER: (Listener Table Queue type)
FALSE
TQMERGE : (Merging Table Queue flag)
FALSE
TQREAD : (Table Queue Read type)
READ AHEAD
TQSEND : (Table Queue Write type)
BROADCAST
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

2) From Operator #5

Estimated number of rows: 144000
Partition Map ID: -100
Partitioning: (COORDINATOR)
Coordinator Partition
Number of columns: 7
Subquery predicate ID: Not Applicable

Column Names:

+Q1.C_COMMENT+Q1.C_ACCTBAL+Q1.C_PHONE
+Q1.C_ADDRESS+Q1.C_NAME+Q1.C_NATIONKEY
+Q1.C_CUSTKEY

Partition Column Names:

+NONE

Output Streams:

3) To Operator #3

Estimated number of rows: 144000
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 7
Subquery predicate ID: Not Applicable

Column Names:

+Q1.C_COMMENT+Q1.C_ACCTBAL+Q1.C_PHONE
+Q1.C_ADDRESS+Q1.C_NAME+Q1.C_NATIONKEY
+Q1.C_CUSTKEY

Partition Column Names:

+NONE

5) SHIP : (Ship)

Cumulative Total Cost: 128168
Cumulative CPU Cost: 4.29461e+09
Cumulative I/O Cost: 32173
Cumulative Re-Total Cost: 2176.51
Cumulative Re-CPU Cost: 4.0959e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 25.8328
Cumulative Comm Cost:0
Cumulative First Comm Cost:0
Estimated Bufferpool Buffers: 32173
Remote communication cost:113565

Arguments:

CSERQY : (Remote common subexpression)
FALSE
DSTSEVER: (Destination (ship to) server)
- (NULL).
RMTQTX : (Remote statement)

```

        SELECT AO."C_CUSTKEY", AO."C_NATIONKEY", AO."C_NAME", AO."C_ADDRESS",
        AO."C_PHONE", AO."C_ACCTBAL", AO."C_COMMENT" FROM "IITEST"."CUSTOMER" AO WHERE
        (AO."C_MKTSEGMENT" = 'AUTOMOBILE') AND (20 < AO."C_NATIONKEY")
        SRCSEVER: (Source (ship from) server)
        ORASERV
        STREAM : (Remote stream)
        FALSE

```

Input Streams:

1) From Object ORA.CUSTOMER

```

        Estimated number of rows: 1.5e+06
        Partition Map ID: -100
        Partitioning:      (COOR )
                        Coordinator Partition
        Number of columns: 9
        Subquery predicate ID: Not Applicable

```

Column Names:

```

+Q1.$RID$+Q1.C_COMMENT+Q1.C_ACCTBAL+Q1.C_PHONE
+Q1.C_ADDRESS+Q1.C_NAME+Q1.C_NATIONKEY
+Q1.C_MKTSEGMENT+Q1.C_CUSTKEY

```

Partition Column Names:

+NONE

Output Streams:

2) To Operator #4

```

        Estimated number of rows: 144000
        Partition Map ID: -100
        Partitioning:      (COOR )
                        Coordinator Partition
        Number of columns: 7
        Subquery predicate ID: Not Applicable

```

Column Names:

```

+Q1.C_COMMENT+Q1.C_ACCTBAL+Q1.C_PHONE
+Q1.C_ADDRESS+Q1.C_NAME+Q1.C_NATIONKEY
+Q1.C_CUSTKEY

```

Partition Column Names:

+NONE

7) FETCH : (Fetch)

Cumulative Total Cost: 53.8812
Cumulative CPU Cost: 200124
Cumulative I/O Cost: 17.5276
Cumulative Re-Total Cost: 43.4524
Cumulative Re-CPU Cost: 145972
Cumulative Re-I/O Cost: 15.5276
Cumulative First Row Cost: 31.2557
Cumulative Comm Cost:0
Cumulative First Comm Cost:0
Estimated Bufferpool Buffers: 18.0566

Arguments:

JN INPUT: (Join input leg)
INNER
MAX RIDS: (Maximum RIDs per list prefetch request)
512
MAXPAGES: (Maximum pages for prefetch)
3
PREFETCH: (Type of Prefetch)
LIST
ROWLOCK : (Row Lock intent)
NEXT KEY SHARE
TABLOCK : (Table Lock intent)
INTENT SHARE
TBISOLVL: (Table access Isolation Level)
CURSOR STABILITY

Predicates:

4) Sargable Predicate
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 5.06148e-07

Predicate Text:

(Q2.O_CUSTKEY = Q1.C_CUSTKEY)

Input Streams:

7) From Operator #8

Estimated number of rows: 5.0634

Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$(A)

Partition Column Names:

+1: Q2.0_ORDERKEY

8) From Object TPCD.ORDERS

Estimated number of rows: 1.00038e+07
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 9
Subquery predicate ID: Not Applicable

Column Names:

+Q2.0_COMMENT+Q2.0_SHIPPRIORITY+Q2.0_CLERK
+Q2.0_ORDERPRIORITY+Q2.0_ORDERDATE
+Q2.0_TOTALPRICE+Q2.0_ORDERSTATUS
+Q2.0_ORDERKEY+Q2.0_CUSTKEY

Partition Column Names:

+1: Q2.0_ORDERKEY

Output Streams:

9) To Operator #3

Estimated number of rows: 5.0634
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 10
Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$+Q2.0_COMMENT+Q2.0_SHIPPRIORITY

```
+Q2.0_CLERK+Q2.0_ORDERPRIORITY+Q2.0_ORDERDATE
+Q2.0_TOTALPRICE+Q2.0_ORDERSTATUS
+Q2.0_ORDERKEY+Q2.0_CUSTKEY
```

Partition Column Names:

```
-----
+1: Q2.0_ORDERKEY
```

```
8) RIDSCN: (Row Identifier Scan)
Cumulative Total Cost: 20.8488
Cumulative CPU Cost: 91759.9
Cumulative I/O Cost: 2
Cumulative Re-Total Cost: 10.4222
Cumulative Re-CPU Cost: 41857
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 20.8481
Cumulative Comm Cost:0
Cumulative First Comm Cost:0
Estimated Bufferpool Buffers: 3
```

Arguments:

```
-----
NUMROWS : (Estimated number of rows)
6
```

Input Streams:

```
-----
6) From Operator #9
```

```
Estimated number of rows: 5.0634
Partition Map ID: 4
Partitioning: (MULT )
Multiple Partitions
Number of columns: 1
Subquery predicate ID: Not Applicable
```

Column Names:

```
-----
+Q2.$RID$(A)
```

Partition Column Names:

```
-----
+1: Q2.0_ORDERKEY
```

Output Streams:

```
-----
7) To Operator #7
```

Estimated number of rows: 5.0634
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$(A)

Partition Column Names:

+1: Q2.0_ORDERKEY

9) SORT : (Sort)

Cumulative Total Cost: 20.8481
Cumulative CPU Cost: 90472.9
Cumulative I/O Cost: 2
Cumulative Re-Total Cost: 10.4208
Cumulative Re-CPU Cost: 39054.2
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 20.8481
Cumulative Comm Cost:0
Cumulative First Comm Cost:0
Estimated Bufferpool Buffers: 3

Arguments:

DUPLWARN: (Duplicates Warning flag)
TRUE
NUMROWS : (Estimated number of rows)
6
ROWWIDTH: (Estimated width of rows)
12
SORTKEY : (Sort Key column)
1: Q2.\$RID\$(A)
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
TRUE

Input Streams:

5) From Operator #10

Estimated number of rows: 5.0634

Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.0_CUSTKEY(A)+Q2.\$RID\$

Partition Column Names:

+1: Q2.0_ORDERKEY

Output Streams:

6) To Operator #8

Estimated number of rows: 5.0634
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$(A)

Partition Column Names:

+1: Q2.0_ORDERKEY

10) IXSCAN: (Index Scan)
Cumulative Total Cost: 20.8465
Cumulative CPU Cost: 87515.2
Cumulative I/O Cost: 2
Cumulative Re-Total Cost: 10.4208
Cumulative Re-CPU Cost: 39054.2
Cumulative Re-I/O Cost: 1
Cumulative First Row Cost: 20.8419
Cumulative Comm Cost: 0
Cumulative First Comm Cost: 0
Estimated Bufferpool Buffers: 3

Arguments:

MAXPAGES: (Maximum pages for prefetch)
 1
 PREFETCH: (Type of Prefetch)
 NONE
 ROWLOCK : (Row Lock intent)
 NONE
 SCANDIR : (Scan Direction)
 FORWARD
 TABLOCK : (Table Lock intent)
 INTENT NONE

Predicates:

 4) Start Key Predicate
 Relational Operator: Equal (=)
 Subquery Input Required: No
 Filter Factor: 5.06148e-07

Predicate Text:

 (Q2.0_CUSTKEY = Q1.C_CUSTKEY)

4) Stop Key Predicate
 Relational Operator: Equal (=)
 Subquery Input Required: No
 Filter Factor: 5.06148e-07

Predicate Text:

 (Q2.0_CUSTKEY = Q1.C_CUSTKEY)

Input Streams:

 4) From Object TPCD.0_CK

Estimated number of rows: 1.00038e+07
 Partition Map ID: 4
 Partitioning: (MULT)
 Multiple Partitions
 Number of columns: 2
 Subquery predicate ID: Not Applicable

Column Names:

 +Q2.0_CUSTKEY(A)+Q2.\$RID\$

Partition Column Names:

+1: Q2.0_ORDERKEY

Output Streams:

5) To Operator #9

Estimated number of rows: 5.0634
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.0_CUSTKEY(A)+Q2.\$RID\$

Partition Column Names:

+1: Q2.0_ORDERKEY

Objects Used in Access Plan:

Schema: TPCD

Name: O_CK

Type: Index

Time of creation: 2004-06-04-07.02.17.390759

Last statistics update: 2004-06-09-23.20.10.138687

Number of columns: 1

Number of rows: 10003789

Width of rows: -1

Number of buffer pool pages: 296184

Distinct row values: No

Tablespace name: INDEX_TS

Tablespace overhead: 9.500000

Tablespace transfer rate: 0.900000

Source for statistics: Single Node

Prefetch page count: 48

Container extent page count: 16

Index clustering statistic: 0.000046

Index leaf pages: 18875

Index tree levels: 3

Index full key cardinality: 1975707

Index first key cardinality: 1975707

Index first 2 keys cardinality: -1

Index first 3 keys cardinality: -1

Index first 4 keys cardinality: -1
Index sequential pages: 18874
Index page density: 99
Index avg sequential pages: 18874
Index avg gap between sequences:0
Index avg random pages: 0
Fetch avg sequential pages: -1
Fetch avg gap between sequences:-1
Fetch avg random pages: -1
Index RID count: 10003789
Index deleted RID count: 0
Index empty leaf pages: 0
Base Table Schema: TPCD
Base Table Name: ORDERS
Columns in index:
O_CUSTKEY

Schema: ORA
Name: CUSTOMER
Type: Nickname
Time of creation: 2004-06-10-08.32.16.950465
Last statistics update:
Number of columns: 8
Number of rows: 1500000
Width of rows: 236
Number of buffer pool pages: 32173
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: TPCD
Name: ORDERS
Type: Table
Time of creation: 2004-06-04-07.02.15.736633
Last statistics update: 2004-06-09-23.20.10.138687
Number of columns: 9
Number of rows: 10003789
Width of rows: 115
Number of buffer pool pages: 296184
Distinct row values: No
Tablespace name: DATA_TS
Tablespace overhead: 9.500000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 48

Container extent page count: 16
Table overflow record count: 0
Table Active Blocks: -1

Archived

Capacity planning in an existing DB2 II environment

In this chapter, we describe a procedure for performing capacity planning in an environment that currently has DB2 Information Integrator (DB2 II) installed. The procedure discusses determining unit cost per query, monitoring the rate of growth of individual queries, defining profiles for categorizing new applications, and computing CPU capacity and memory requirements for a projected query workload.

The topics covered are:

- ▶ Capacity planning assumptions
- ▶ Capacity planning procedure
- ▶ Capacity planning new applications

5.1 Introduction

Capacity planning in a federated server environment is a challenging task given the complexity of possible configurations, and the autonomous nature of the remote heterogeneous data sources involved. For example, the federated server may be a dedicated system (no local data), or a shared system with local data that may be a single partition (non-DPF) or multiple partition environment (DPF). Add to that the multiplicity of operating system platforms supported by DB2 II, and the impact on federated server performance by throughput capabilities of the remote data sources, we have a difficult challenge in capacity planning.

Then there is the traditional challenge of projecting the exact workload for which capacity planning must be undertaken, a task considerably more difficult for customer environments that do not have DB2 II currently installed.

In the following sections, we discuss our capacity planning assumptions, and propose a procedure for performing capacity planning in an existing DB2 II environment, which could then possibly be adapted for more complex environments.

5.2 Capacity planning assumptions

Our objective here is to document a capacity planning procedure for existing DB2 II customer environments based on the facilities and tools available to us, with the expectation that the reader would then improvise on the process to suit her particular environment.

We make the following assumptions about the capacity planning procedure documented here:

- ▶ We focus on CPU and memory utilization only, and not on the capacity requirements of the I/O subsystem.
- ▶ The federated server is on a dedicated AIX system. There is no local data access.
- ▶ A significant portion of the query workload is dynamic SQL.
- ▶ There are no significant changes to the system environment or access paths for the query workload. In practice, you would need to keep track of these changes to ensure that only the appropriate measured metrics for each significant query are used for capacity planning.
- ▶ We assume that the SET SERVER OPTION was not used for the federated queries. When this statement is issued, the server option changes are temporary and are not recorded in the global catalog. However, when we

EXPLAIN the user queries found in the dynamic cache, the server options correspond to the settings in the global catalog, which may not accurately reflect the actual access paths generated and executed.

- ▶ This procedure does not describe any sophisticated algorithms to use in selecting the appropriate reporting interval(s) for effective sizing; for example, one needs to filter out the reporting intervals that have outliers of utilizations, and only focus on the median, average, or second-highest utilizations, depending upon the degree of risk tolerated in sizing estimates.
- ▶ The accuracy of the procedure depends upon gathering accurate metrics over a representative interval (large enough sample), and being able to project a future workload that includes the number of concurrent users, the frequency of existing queries, as well as the profile and frequency of new queries.

Attention: We ran a number of controlled measurements in our P650 8-way 32 GB AIX 5.2 environment using select queries against 10 GB of TPCD data at remote DB2, Oracle and SQL Server data sources, and found our estimates to be within a couple percent of actual utilization.

5.3 Capacity planning procedure

It is important to understand the following underpinnings of the capacity planning procedure in order to appreciate its strengths and limitations and evaluate its efficacy in your particular environment.

- ▶ Dynamic SQL snapshot information in the dynamic cache contains, along with other information, accurate metrics about the text of each query executed by the user, the number of executions, the total user and system CPU time, the total elapsed time, and the number of rows returned to the user.

The dynamic cache also contains the SQL fragment executed at the remote data source. However, the metrics of particular relevance to this SQL fragment are the SQL text, the number of rows returned (to the federated server), the number of executions, and the total elapsed time. A limitation is that the total CPU time is always zero, and there is no direct mechanism of relating this SQL fragment to the user's query.

This information is used to:

- Determine the unit CPU cost of each query.
- Project the frequency of query execution in future.
- Define a profile of queries based on the number and type of data sources accessed and the number of rows returned from each data source.

- ▶ The **db2 get snapshot for dynamic sql for <dbname>** command for listing the contents of the dynamic cache, or the use of the equivalent SQL table function ... FROM TABLE(SNAPSHOT_DYN_SQL(", -2))
- ▶ The DB2 EXPLAIN facility's **db2exfmt** command for explaining the SQL statement query text in the dynamic cache.

It is used for establishing the relationship between the user query and the SQL fragment in the dynamic cache. More on this later in 5.3.1, "Capacity planning procedure overview" on page 381.

- ▶ The **db2 get snapshot for db on <dbname>** command is used to:
 - Determine the high water mark for concurrent connections to the federated server.
 - Determine shared sort heap utilization for memory estimation.
- ▶ The **db2 get snapshot for dbm** command is used to determine memory utilization of the private sort heap and other heaps.
- ▶ The **db2batch** utility is used in special situations when accurate metrics are not available for a given query from the dynamic cache.
- ▶ The Event Monitor can provide very detailed information on each operation, such as elapsed time and CPU utilization by query. However, it currently does not provide information about access to remote data sources, and its overhead at peak intervals may not be acceptable. It is primarily a problem determination tool.
- ▶ The **sar** operating system's command to obtain the average CPU utilization during the monitoring interval. This is used to establish the capture ratio for the monitored workload.

The capture ratio represents the ratio of CPU utilization as measured by the dynamic cache metrics and the CPU utilization measured by the operating system's **sar** command. The difference varies by the nature of the workload and represents other DB2 and system processes' CPU consumption not captured in the dynamic cache metrics, but should be factored into sizing the CPU for a projected workload. More on this later in 5.3.1, "Capacity planning procedure overview" on page 381.

Attention: Our controlled measurements in a dedicated federated server environment show very little difference between operating system measured CPU utilization and that obtained from the dynamic cache. However, this may not apply in your environment, and may be worth evaluating in your own environment.

- ▶ The **vmstat** operating system's command to obtain the high water mark memory utilization.

5.3.1 Capacity planning procedure overview

Figure 5-1 on page 382 provides an overview of the steps involved, as follows:

1. Step 1: Establish environment.

This step involves setting the appropriate DB2 monitor switches, creating the EXPLAIN tables, creating a performance warehouse to store the snapshot results of the dynamic cache, and summary results of utilization.

2. Step 2: Capture runtime metrics.

This step involves choosing the representative monitoring interval, capturing the contents of the dynamic cache at representative intervals, capturing the operating system measure of CPU and memory during these intervals, explaining the SQL statements in the dynamic cache, and establishing the relationship between remote SQL fragments in the user query.

3. Step 3: Summarize monitored intervals information.

This step involves summarizing the information about user queries, and their corresponding query fragments and EXPLAIN output for each monitored interval. This includes the CPU times, elapsed times and number of rows processed.

4. Step 4: Identify reporting interval.

This step involves identifying which of the various runtime metrics captured in the previous step are most relevant to computing the unit CPU cost and memory utilization for the workload.

5. Step 5: Generate the utilization report.

This step involves generating the unit CPU cost per query, total memory utilization, and projected growth rate of workload from the reporting interval identified in the previous step.

6. Step 6: Estimate capacity for the anticipated future growth.

This step involves a spread sheet computation of identifying the queries in a future workload, their frequency of execution, and throughput in order to estimate CPU and memory capacity requirements for the anticipated workload.

Each of these steps is elaborated on in the following sections.

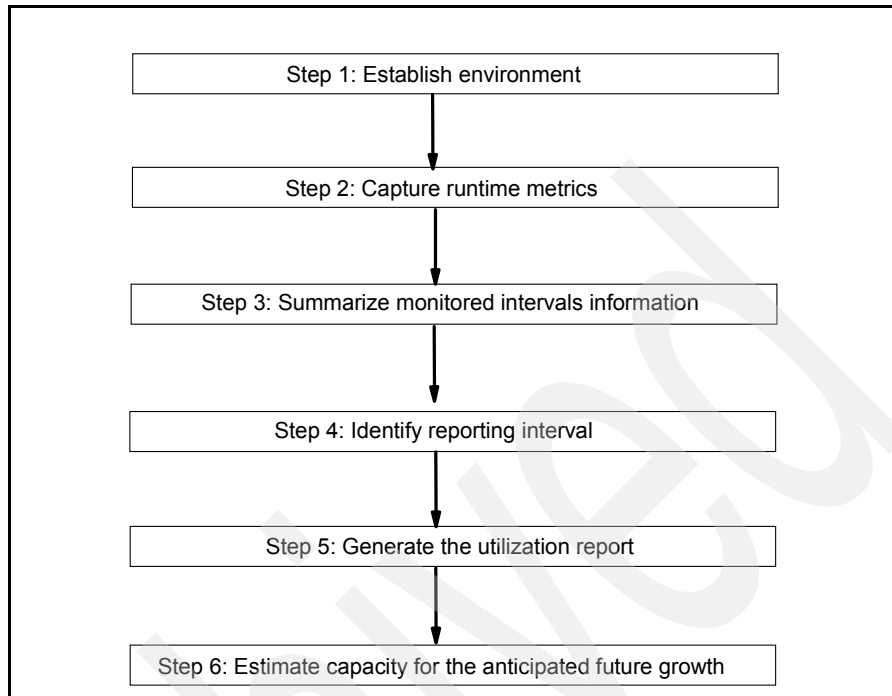


Figure 5-1 Capacity planning procedure overview

Step 1: Establish environment

The following items need to be established before ongoing capacity planning related monitoring can begin:

1. Set the DB2 monitor switches.

In order to gather the dynamic SQL, buffer pool, and sort memory utilization information in the snapshot monitor, the DFT_MON_STMT, DFT_MON_BUFPOOL, and DFT_MON_SORT switches must be set as shown in Example 5-1.

Example 5-1 Set the DB2 monitor switches

```
$ db2 -t update dbm cfg using dft_mon_stmt on;
DB20000I The UPDATE DATABASE MANAGER CONFIGURATION command completed
successfully.
$ db2 -t update dbm cfg using dft_mon_bufpool on;
DB20000I The UPDATE DATABASE MANAGER CONFIGURATION command completed
successfully.
$ db2 -t update dbm cfg using dft_mon_sort on;
```

DB20000I The UPDATE DATABASE MANAGER CONFIGURATION command completed successfully.

2. Create the EXPLAIN tables.

First create a user for the performance warehouse, and then create the EXPLAIN tables for that user's schema. The EXPLAIN tables are created using the EXPLAIN.DDL script in the **\$HOME/sqllib/misc/** directory, as shown in Example 5-2, where **\$HOME** is the DB2/II instance owner's home directory. On Windows, the EXPLAIN.DDL script can be found in **c:\Program files\IBM\SQLLIB\misc**. EXPLAIN tables are used for determining the relationship between the user query and remote SQL fragments found in the dynamic cache.

Example 5-2 Create the EXPLAIN tables

```
$db2 -tvf $HOME/sqllib/misc/EXPLAIN.DDL
```

Note: The EXPLAIN.DDL script file creates EXPLAIN tables in the default user table space (usually USERSPACE1 if the defaults were taken when the database was created). You should modify this script to create the EXPLAIN tables in the desired table space.

3. Create the performance warehouse tables.

There are a number of tables to be created in the performance warehouse, as shown in Example 5-3 on page 384.

- CREATE SEQUENCE FEDWH.GENSNAPID creates a sequence of name FEDWH.GENSNAPID. It is used in generating keys for the start and end of a monitoring interval.
- FEDWH.FEDWH_INSTANCE table that contains the average memory utilization, maximum connections, operating system recorded CPU (via **sar** command), and the computed capture ratio during each monitored interval.
- FEDWH.FEDWH_SNAPSHOT_DYN_SQL table containing the snapshot of dynamic cache with an additional "SNAPID" column.
- FEDWH.FEDWH_EXPLAIN_INSTANCE table that stores the relationship between the explain table contents and the snapshot interval.
- FEDWH.FEDWH_SNAPSHOT_DYN_SQL_INTERVAL table containing metrics of the dynamic cache for a particular monitoring interval.
- FEDWH.FEDWH_FEDSQL_INTERVAL table containing combined metrics of the dynamic cache and explain table contents for each query in a particular monitoring interval.

- FEDWH.FEDWH_INSTANCE_REPORT and FEDWH.FEDWH_FEDSQL_REPORT are tables that contain the utilization reports.

Example 5-3 Create performance warehouse tables

```

-----
--Create a sequence at the application server.
-----
CREATE SEQUENCE FEDWH.GENSNAPID;
-----
--Create FEDWH.FEDWH_INSTANCE table contains the average memory utilization,
--maximum connections, operating system recorded CPU (via sar command), and the
--computed capture ratio during each monitored interval.
-----
CREATE TABLE FEDWH.FEDWH_INSTANCE (
  S_SNAPID          INTEGER,
  E_SNAPID          INTEGER,
  S_SNAPSHOT_TIMESTAMP  TIMESTAMP,
  E_SNAPSHOT_TIMESTAMP  TIMESTAMP,
  SAR_CPU           DECIMAL(4,1),
  CAPTURE_RATIO     DECIMAL(6,3),
  MEMORYCONSUMED    INTEGER,
  MAXIMUMCONNECTIONS  INTEGER
) IN FEDWH_DTS1;
-----
--Create FEDWH.FEDWH_SNAPSHOT_DYN_SQL table containing a snapshot of the
-- dynamic cache
-----
CREATE VIEW FEDWH.VTMP_SNAPSHOT_DYN_SQL
  AS SELECT CAST(1 AS INTEGER) AS SNAPID ,DYNSQL.*
  FROM TABLE(SNAPSHOT_DYN_SQL('','-2)) AS DYNSQL;
--
CREATE TABLE FEDWH.FEDWH_SNAPSHOT_DYN_SQL
  LIKE FEDWH.VTMP_SNAPSHOT_DYN_SQL IN FEDWH_DTS1;
--Drop Dummy VIEW
DROP VIEW FEDWH.VTMP_SNAPSHOT_DYN_SQL;
-----
--Create FEDWH.FEDWH_EXPLAIN_INSTANCE table to store the relationship between
--the explain table contents and the monitoring interval it relates to
-----
CREATE VIEW FEDWH.VTMP_EXPLAIN_INSTANCE
  AS SELECT CAST(1 AS INTEGER) AS SNAPID,
  T1.EXPLAIN_REQUESTER, T1.EXPLAIN_TIME,T1.SOURCE_NAME,
  T1.SOURCE_SCHEMA,T1.SOURCE_VERSION
  FROM EXPLAIN_INSTANCE AS T1;

CREATE TABLE FEDWH.FEDWH_EXPLAIN_INSTANCE

```

```

LIKE FEDWH.VTMP_EXPLAIN_INSTANCE IN FEDWH_DTS1;

ALTER TABLE FEDWH.FEDWH_EXPLAIN_INSTANCE ADD CONSTRAINT PK_FW_EXPINST
PRIMARY KEY
(EXPLAIN_REQUESTER, EXPLAIN_TIME, SOURCE_NAME, SOURCE_SCHEMA, SOURCE_VERSION);
--Drop Dummy VIEW
DROP VIEW FEDWH.VTMP_EXPLAIN_INSTANCE;
-----
--Create FEDWH.FEDWH_SNAPSHOT_DYN_SQL_INTERVAL table that contains the
--metrics of the dynamic cache for a particular monitoring interval
-----
CREATE VIEW FEDWH.VTMP_SNAPSHOT_DYN_SQL_INTERVAL
AS SELECT T1.S_SNAPID, T1.E_SNAPID, T1.S_SNAPSHOT_TIMESTAMP,
T1.E_SNAPSHOT_TIMESTAMP, T2.ROWS_READ,
T2.ROWS_WRITTEN, T2.NUM_EXECUTIONS, T2.NUM_COMPILATIONS,
T2.PREP_TIME_WORST, T2.PREP_TIME_BEST, T2.INT_ROWS_DELETED,
T2.INT_ROWS_INSERTED, T2.INT_ROWS_UPDATED,
T2.STMT_SORTS, T2.TOTAL_EXEC_TIME, T2.TOTAL_SYS_CPU_TIME,
T2.TOTAL_USR_CPU_TIME, T2.STMT_TEXT
FROM FEDWH.FEDWH_INSTANCE AS T1, TABLE(SNAPSHOT_DYN_SQL(' ', -2)) AS T2 ;
--
CREATE TABLE FEDWH.FEDWH_SNAPSHOT_DYN_SQL_INTERVAL
LIKE FEDWH.VTMP_SNAPSHOT_DYN_SQL_INTERVAL IN FEDWH_DTS1;
--Drop Dummy VIEW
DROP VIEW FEDWH.VTMP_SNAPSHOT_DYN_SQL_INTERVAL;
-----
--Create FEDWH.FEDWH_FEDSQL_INTERVAL table that contains the combined
--metrics of the dynamic cache and explain table contents for each query in a
--particular monitoring interval
-----
CREATE TABLE FEDWH.FEDWH_FEDSQL_INTERVAL(
S_SNAPID          INTEGER,
E_SNAPID          INTEGER,
STATEMENT_TEXT    CLOB,
NUM_EXECUTIONS    BIGINT,
AVG_CPU_TIME      DECIMAL(31,12),
--if required in future -- AVG_STMT_SORTS    DECIMAL(31,1),
SRCSEVER          VARCHAR(8),
RMTQTX           CLOB,
--if required in future -- RM_AVG_EXEC_TIME DECIMAL(31,12),
RM_AVG_NUM_EXEC   DECIMAL(31,1),
RM_AVG_ROWS       DECIMAL(31,2)
) IN FEDWH_DTS1;
-----
-----
--Create FEDWH.FEDWH_INSTANCE_REPORT and FEDWH.FEDWH_FEDSQL_REPORT that
--contains the utilization reports at various times

```

```

CREATE TABLE FEDWH.FEDWH_INSTANCE_REPORT (
  REPORT_ID          INTEGER,
  REPORT_DATE        DATE,
  TOTAL_INTERVAL     INTEGER,
  SAR_CPU            DECIMAL(4,1),
  CAPTURE_RATIO      DECIMAL(4,3),
  MEMORYCONSUMED     INTEGER,
  MAXIMUMCONNECTIONS INTEGER
) IN FEDWH_DTS1;

CREATE TABLE FEDWH.FEDWH_FEDSQL_REPORT(
  REPORT_ID          INTEGER,
  STATEMENT_TEXT     CLOB,
  NUM_EXECUTIONS     BIGINT,
  EXEC_PER_SEC       DECIMAL(31,10),
  AVG_CPU_TIME       DECIMAL(31,12),
  -- if required in future -- AVG_EXEC_TIME    DECIMAL(31,12),
  AVG_STMT_SORTS     DECIMAL(31,1),
  SRCSEVER           VARCHAR(8),
  RMTQTX             CLOB,
  -- if required in future -- RM_AVG_EXEC_TIME DECIMAL(31,12),
  RM_AVG_NUM_EXEC    DECIMAL(31,1),
  RM_AVG_ROWS        DECIMAL(31,2)
) IN FEDWH_DTS1;

```

4. Rebind the tools used.

The isolation levels of the **db2batch** (default RR) and **db2exfmt** (default CS) commands should match the isolation level of the SQL query being processed.

Example 5-4 shows an example of binding **db2batch** with isolation level of cursor stability (CS), while Example 5-5 shows **db2exfmt** being bound with isolation level uncommitted read (UR).

Example 5-4 Bind db2batch with CS isolation level

```

$ cd $HOME/sql1lib/bnd
$ db2 bind db2batch.bnd blocking all grant public isolation cs

```

Example 5-5 DBM CFG parameter settings affecting connections

```

$ cd $HOME/sql1lib/bnd
$ db2 bind db2exfmt.bnd blocking all grant public isolation ur

```

Step 2: Capture runtime metrics

Once the capacity planning environment has been established, runtime metrics need to be collected on an ongoing basis during the day over an extended period

of time (months) at intervals that represent heavy or peak processing on the system.

Each organization tends to have different representative intervals, and it is up to the database administrator to identify them in their organization. For example, the weekday representative intervals may be between 10 a.m. and 11 a.m, and 4 p.m. and 5 p.m, while the weekend representative intervals may be between 2 p.m. and 3 p.m.

The following steps must be followed for each representative interval:

1. No package cache overflows.

Ensure that the contents of the dynamic cache are in a steady state with very few or no overflows, so that information about queries executing during the representative interval are not lost. Example 5-6 shows the snapshot command to determine the state of the dynamic cache.

Example 5-6 Checking the state of the package cache

```
$ db2 get snapshot for db on fedserv|grep "Package cache"
Package cache lookups           = 71321
Package cache inserts          = 18
Package cache overflows         = 0
Package cache high water mark (Bytes) = 294977
$ db2 get db cfg for fedserv|grep -e "MAXAPPLS" -e "PCKCACHESZ"
Catalog cache size (4KB)       (CATALOGCACHE_SZ) = (MAXAPPLS*4)
Package cache size (4KB)       (PCKCACHESZ) = (MAXAPPLS*8)
Max number of active applications (MAXAPPLS) = 200
```

The key field to check is the **Package cache overflows** field, which should ideally be zero. If not, tune the package cache (database configuration parameter PCKCACHESZ) to avoid overflows. Refer to the IBM Redbook *DB2 UDB ESE V8 non-DPF Performance Guide for High Performance OLTP and BI*, SG24-6432, for details on tuning PCKCACHESZ.

2. Issue a dynamic SQL snapshot at the start of the monitoring interval.

The objective is to capture details about all the queries in the dynamic cache and store it in the FEDWH.FEDWH_SNAPSHOT_DYN_SQL performance warehouse table with an identifier in the SNAPID column, as shown in Example 5-7.

Example 5-7 Capture dynamic SQL snapshot into performance warehouse table

```
VALUES(NEXTVAL FOR FEDWH.GENSNAPID);
INSERT INTO FEDWH.FEDWH_SNAPSHOT_DYN_SQL SELECT PREVVAL FOR
FEDWH.GENSNAPID,DYNSQL.* FROM TABLE(SNAPSHOT_DYN_SQL('','-2')) AS DYNSQL WHERE
NUM_EXECUTIONS>0;
```

The NEXTVAL expression generates and returns the next value for the sequence FEDWH.GENSNAPID. The predicate NUM_EXECUTIONS > 0 is meant to exclude dynamic cache entries that have no successful executions.

Note: Every start monitoring interval SNAPID value would have to be different. It is used as an identifier for the start and end of a monitoring interval.

Example 5-8 is an example of snapshot output using the `get snapshot for dynamic sql ...` command. Some of the key fields are highlighted.

Example 5-8 Dynamic SQL snapshot

```
$db2 get snapshot for dynamic sql on fedserv

      Dynamic SQL Snapshot Result

Database name           = FEDSERV
Database path           = /data1/kawa/kawa/NODE0000/SQL00001/
.....lines have been removed.....

Number of executions    = 13826
Number of compilations  = 1
Worst preparation time (ms) = 44
Best preparation time (ms) = 44
Internal rows deleted   = 0
Internal rows inserted  = 0
Rows read               = 0
Internal rows updated   = 0
Rows written            = 0
Statement sorts         = 27644
Statement sort overflows = 0
Total sort time         = 13823
Buffer pool data logical reads = 0
Buffer pool data physical reads = 0
Buffer pool temporary data logical reads = 0
Buffer pool temporary data physical reads = 0
Buffer pool index logical reads = 0
Buffer pool index physical reads = 0
Buffer pool temporary index logical reads = 0
Buffer pool temporary index physical reads = 0
Total execution time (sec.ms) = 4601.005432
Total user cpu time (sec.ms) = 780.960000
Total system cpu time (sec.ms) = 160.310000
Statement text          = SELECT 'Q03_q3',           L_ORDERKEY,
SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE,    O_ORDERDATE,
```

```
O_SHIPRIORITY FROM      db2.CUSTOMER,      db2.ORDERS,      ora.LINEITEM
WHERE      C_CUSTKEY BETWEEN 1 AND 100 AND      C_CUSTKEY = O_CUSTKEY AND
L_ORDERKEY = O_ORDERKEY AND      O_ORDERDATE < DATE('1995-03-15') AND
L_SHIPDATE > DATE('1995-03-15') GROUP BY      L_ORDERKEY,      O_ORDERDATE,
O_SHIPRIORITY ORDER BY      REVENUE DESC,      O_ORDERDATE FETCH FIRST 10
ROWS ONLY
```

.....lines have been removed.....

```
Number of executions      = 13826
Number of compilations    = 13826
Worst preparation time (ms)      = 0
Best preparation time (ms)      = 0
Internal rows deleted      = 0
Internal rows inserted      = 0
Rows read      = 1686501
Internal rows updated      = 0
Rows written      = 0
Statement sorts      = 0
Buffer pool data logical reads      = 0
Buffer pool data physical reads      = 0
Buffer pool temporary data logical reads      = 0
Buffer pool temporary data physical reads      = 0
Buffer pool index logical reads      = 0
Buffer pool index physical reads      = 0
Buffer pool temporary index logical reads      = 0
Buffer pool temporary index physical reads      = 0
Total execution time (sec.ms)      = 57.505651
Total user cpu time (sec.ms)      = 0.000000
Total system cpu time (sec.ms)      = 0.000000
Statement text      = [ORASERV] SELECT A0."L_ORDERKEY",
A0."L_EXTENDEDPRICE", A0."L_DISCOUNT" FROM "IITEST"."LINEITEM" A0 WHERE
(A0."L_ORDERKEY" = :H0 ) AND (TO_DATE('19950315 000000','YYYYMMDD HH24MISS') <
A0."L_SHIPDATE") ORDER BY 1 ASC, A0."L_SHIPDATE" ASC
```

.....lines have been removed.....

```
Number of executions      = 13822
Number of compilations    = 13822
Worst preparation time (ms)      = 0
Best preparation time (ms)      = 0
Internal rows deleted      = 0
Internal rows inserted      = 0
Rows read      = 6330476
Internal rows updated      = 0
Rows written      = 0
Statement sorts      = 0
```

```

Buffer pool data logical reads      = 0
Buffer pool data physical reads    = 0
Buffer pool temporary data logical reads = 0
Buffer pool temporary data physical reads = 0
Buffer pool index logical reads    = 0
Buffer pool index physical reads    = 0
Buffer pool temporary index logical reads = 0
Buffer pool temporary index physical reads = 0
Total execution time (sec.ms)      = 108.711819
Total user cpu time (sec.ms)      = 0.000000
Total system cpu time (sec.ms)    = 0.000000
Statement text                      = [DB2SERV] SELECT A1."O_ORDERKEY",
A1."O_ORDERDATE", A1."O_SHIPPRIORITY" FROM "TPCD"."CUSTOMER" AO,
"TPCD"."ORDERS" A1 WHERE (1 <= AO."C_CUSTKEY") AND (AO."C_CUSTKEY" <= 100) AND
(A1."O_ORDERDATE" < '1995-03-15') AND (A1."O_CUSTKEY" <= 100) AND (1 <=
A1."O_CUSTKEY") AND (AO."C_CUSTKEY" = A1."O_CUSTKEY") FOR READ ONLY

.....lines have been removed.....

```

Note: Example 5-8 shows both user-entered SQL statements, as well as remote SQL fragments, which include the server name in front of the text like “[ORASERV] SELECT A?.”colname”....FROM...”.

Example 5-8 on page 388 executes the user-entered query that joins data in DB2 and Oracle data sources 13826 times.

There are two remote SQL fragments that also execute 13826 and 13822 times, respectively. The user-entered SQL consumed 0.246 CPU seconds per query, which is $((\text{Total user cpu time (sec.ms)} + \text{Total system cpu time (sec.ms)}) / (\text{Number of executions}))$. The DB2 data source returned 458 rows per query, which is $((\text{Rows read}) / (\text{Number of executions}))$, while the Oracle data source returned 121.98 rows per query.

Note: The **Total user cpu time (sec.ms)** and **Total system cpu time (sec.ms)** fields for the remote SQL fragment are always zero.

3. Issue the operating system **sar** command for the monitoring interval.

The objective is to capture CPU utilization as monitored by the operating system for the monitoring interval, as shown in Example 5-9 on page 391.

Example 5-9 Output of the operating system sar command

```
bash-2.05b# sar -o tmp.sar 60 10
```

```
AIX jamesbay 2 5 00016DFA4C00 08/01/04
```

	%usr	%sys	%wio	%idle
19:12:26	13	5	0	82
19:13:26	12	5	0	83
19:14:26	13	5	0	82
19:15:26	13	5	0	82
19:16:26	12	5	0	82
19:17:26	13	5	0	82
19:18:26	12	5	0	82
19:19:26	13	5	0	82
19:20:26	13	5	0	82
19:21:26	12	5	0	83
19:22:26	13	5	0	82
Average	13	5	0	82

4. Issue the operating system **vmstat** command for the monitoring interval.

The objective is to capture memory utilization as monitored by the operating system for the monitoring interval, as shown in Example 5-10.

Example 5-10 Output of the operating system vmstat command

```
$vmstat -t 10 60 > vmstat.tmp&
```

kthr		memory			page				faults				cpu			time			
r	b	avm	fre	re	pi	po	fr	sr	cy	in	sy	cs	us	sy	id	wa	hr	mi	se
2	1	1331974	7006323	0	0	0	0	0	0	0	1216	3544	1561	1	0	99	0		
19:12:26																			
2	0	1332002	7006293	0	0	0	0	0	0	0	9276	22135	18105	13	5	82	0		
19:12:36																			
2	0	1332002	7006293	0	0	0	0	0	0	0	9268	22211	18123	13	5	83	0		
19:12:46																			
2	0	1332001	7006294	0	0	0	0	0	0	0	9248	22200	18091	13	5	82	0		
19:12:56																			
1	0	1332002	7006293	0	0	0	0	0	0	0	9277	22426	18212	12	5	82	0		
19:13:06																			
1	0	1332002	7006293	0	0	0	0	0	0	0	9349	23695	18348	13	5	82	0		
19:13:16																			
1	0	1332003	7006291	0	0	0	0	0	0	0	9546	22548	18737	12	5	83	0		
19:13:26																			
1	0	1332003	7006291	0	0	0	0	0	0	0	9347	22444	18318	13	5	82	0		
19:13:36																			
1	0	1332003	7006291	0	0	0	0	0	0	0	9293	22626	18279	12	5	82	0		
19:13:46																			
1	0	1332002	7006292	0	0	0	0	0	0	0	9289	22296	18210	12	5	82	0		
19:13:56																			

```

1 0 1332003 7006291 0 0 0 0 0 0 9293 22410 18249 12 5 83 0
19:14:06
2 0 1332003 7006291 0 0 0 0 0 0 9367 23934 18440 12 5 83 0
19:14:16
1 0 1332004 7006289 0 0 0 0 0 0 9579 22762 18826 12 5 83 0
19:14:26
2 0 1332002 7006291 0 0 0 0 0 0 9345 22690 18379 12 5 83 0
19:14:36
1 0 1332004 7006289 0 0 0 0 0 0 9323 22354 18270 12 5 82 0
19:14:46
1 0 1332004 7006289 0 0 0 0 0 0 9307 22375 18224 13 5 82 0
19:14:56
1 0 1332003 7006290 0 0 0 0 0 0 9274 22345 18178 13 6 82 0
19:15:06
1 0 1332004 7006289 0 0 0 0 0 0 9258 22588 18200 13 5 82 0
19:15:16
1 0 1332452 7006235 0 0 0 0 0 0 9263 22699 18244 13 5 82 0
19:15:26
1 0 1332037 7006255 0 0 0 0 0 0 9290 22119 18165 13 5 82 0
19:15:36
kthr      memory          page      faults      cpu      time
.....lines have been removed.....

```

The high water mark can be determined as shown in Example 5-11.

Example 5-11 High water mark value of memory utilization

```

$sed -n '/^ *[0-9]/p' vmstat.tmp|sort -n +3|head -1
1 0 1333548 7004564 0 0 0 0 0 0 9244 22522 18160 12 5 82 0 19:21:56

```

5. Issue a dynamic SQL snapshot at the end of the monitoring interval.

Use the query with the same SNAPID value shown in Example 5-12 to capture information from the dynamic cache. The value of SNAPID is 2 and helps identify the rows in the FEDWH.FEDWH_SNAPSHOT_DYN_SQL table as defining the bounds of this particular monitoring interval.

Example 5-12 Capture dynamic SQL snapshot into performance warehouse table

```

VALUES(NEXTVAL FOR FEDWH.GENSNAPID);
INSERT INTO FEDWH.FEDWH_SNAPSHOT_DYN_SQL SELECT PREVVAL FOR
FEDWH.GENSNAPID,DYNSQL. FROM TABLE(SNAPSHOT_DYN_SQL(' ',-2)) AS DYNSQL WHERE
NUM_EXECUTIONS>0;

```

Figure 5-2 on page 393 through Figure 5-4 on page 395 show the contents of the FEDWH.FEDWH_SNAPSHOT_DYN_SQL table after issuing the SQL statements shown in Example 5-7 on page 387 and Example 5-12.

Command Editor 1						
Command Editor Selected Edit View Tools Help						
Commands Query Results Access Plan						
SNAPID	SNAPSHOT_TIMESTAMP	ROWS_READ	ROWS_WRITTEN	NUM_EXECUTIONS	NUM_COMPILATIONS	
1	2004/08/01 19:12:19 166037	2	0	1451	1	
1	2004/08/01 19:12:19 166037	2	0	1461	1	
1	2004/08/01 19:12:19 166037	0	0	510	1	
1	2004/08/01 19:12:19 166037	5	0	2	1	
1	2004/08/01 19:12:19 166037	1	0	1	1	
1	2004/08/01 19:12:19 166037	2	0	1499	1	
1	2004/08/01 19:12:19 166037	61861	0	232195	509	
1	2004/08/01 19:12:19 166037	22470	0	1498	1498	
1	2004/08/01 19:12:19 166037	1461	0	1461	1461	
1	2004/08/01 19:12:19 166037	231290	0	505	505	
1	2004/08/01 19:12:19 166037	1451	0	1451	1451	
1	2004/08/01 19:12:19 166037	1499	0	1499	1499	
2	2004/08/01 19:22:37 448710	2	0	30766	1	
2	2004/08/01 19:22:37 448710	2	0	30836	1	
2	2004/08/01 19:22:37 448710	0	0	10319	1	
2	2004/08/01 19:22:37 448710	0	0	1	1	
2	2004/08/01 19:22:37 448710	5	12	3	1	
2	2004/08/01 19:22:37 448710	1	0	2	1	
2	2004/08/01 19:22:37 448710	2	0	30692	1	
2	2004/08/01 19:22:37 448710	1258615	0	4724809	10319	
2	2004/08/01 19:22:37 448710	460380	0	30692	30692	
2	2004/08/01 19:22:37 448710	30836	0	30836	30836	
2	2004/08/01 19:22:37 448710	4723354	0	10313	10313	
2	2004/08/01 19:22:37 448710	30766	0	30766	30766	
2	2004/08/01 19:22:37 448710	30692	0	30692	30692	
3	2004/08/01 20:07:05 716725	8	0	1	1	

Displays the results editable table.

Delete Row

Commit Roll Back

Fetch More Rows

☐ Automatically commit updates

100 row(s) in memory

Figure 5-2 Contents of FEDWH.FEDWH_SNAPSHOT_DYN_SQL table (1 of 3)

Command Editor 1							
Command Editor Selected Edit View Tools Help							
Commands Query Results Access Plan							
PREP_TIME_WORST	PREP_TIME_BEST	INT_ROWS_DELETED	INT_ROWS_INSERTED	INT_ROWS_UPDATED	STM	Add Row	Delete Row
51	51	0	0	0	0		
26	26	0	0	0	0		
44	44	0	0	0	0		
35	35	0	0	0	0		
1	1	0	0	0	0		
21	21	0	0	0	0		
0	0	0	0	0	0		
0	0	0	0	0	0		
0	0	0	0	0	0		
0	0	0	0	0	0		
0	0	0	0	0	0		
0	0	0	0	0	0		
0	0	0	0	0	0		
51	51	0	0	0	0		
26	26	0	0	0	0		
44	44	0	0	0	0		
34	34	0	0	0	0		
35	35	0	0	0	0		
1	1	0	0	0	0		
21	21	0	0	0	0		
0	0	0	0	0	0		
0	0	0	0	0	0		
0	0	0	0	0	0		
0	0	0	0	0	0		
0	0	0	0	0	0		
0	0	0	0	0	0		
1	1	0	0	0	0		
						100 row(s) in memory	

Figure 5-3 Contents of FEDWH.FEDWH_SNAPSHOT_DYN_SQL table (2 of 3)

STMT_SORTS	TOTAL_EXEC_TIME	TOTAL_SYS_CPU_TIME	TOTAL_USR_CPU_TIME	STMT_TEXT
0	0	17	0	0 SELECT 'Q00',C_NAME F...
0	0	7	0	0 SELECT 'Q01',C_NAME F...
0	1010	181	5	28 SELECT 'Q03_q3', L_...
0	0	0	0	0 INSERT INTO FEDWH.FED...
0	0	0	0	0 VALUES(NEXTVAL FOR ...
0	0	8	0	1 SELECT 'Q02',C_NAME,O...
0	0	2	0	0 [ORASERV] SELECT A0."...
0	0	1	0	0 [ORASERV] SELECT A0."...
0	0	7	0	0 [SQLSERV] SELECT A0."...
0	0	6	0	0 [DB2SERV] SELECT A1 "...
0	0	2	0	0 [DB2SERV] SELECT A0 "...
0	0	1	0	0 [DB2SERV] SELECT A0 "...
0	0	51	1	10 SELECT 'Q00',C_NAME F...
0	0	162	5	18 SELECT 'Q01',C_NAME F...
0	20626	3436	119	581 SELECT 'Q03_q3', L_...
0	0	0	0	0 SELECT COUNT(*) FROM ...
0	0	0	0	0 INSERT INTO FEDWH.FED...
0	0	0	0	0 VALUES(NEXTVAL FOR ...
0	0	71	2	22 SELECT 'Q02',C_NAME,O...
0	0	42	0	0 [ORASERV] SELECT A0."...
0	0	26	0	0 [ORASERV] SELECT A0."...
0	0	158	0	0 [SQLSERV] SELECT A0."...
0	0	81	0	0 [DB2SERV] SELECT A1 "...
0	0	33	0	0 [DB2SERV] SELECT A0 "...
0	0	33	0	0 [DB2SERV] SELECT A0 "...
0	0	0	0	0 SELECT SRCSEVER ,RM...

Figure 5-4 Contents of FEDWH.FEDWH_SNAPSHOT_DYN_SQL table (3 of 3)

6. Capture the maximum number of connections.

The objective here is to determine the maximum number of concurrent connections at the end of dynamic SQL snapshot capture by issuing another snapshot, as shown in Example 5-13.

Example 5-13 Snapshot for maximum concurrent connections

```
db2 get snapshot for db on fedserv|grep -i conn
First database connect timestamp      = 08/01/2004 19:11:23.452131
High water mark for connections    = 9
Application connects                  = 17
Secondary connects total              = 0
Applications connected currently      = 9
```

Note: The *High water mark for connections* field is actually the highest number of simultaneous connections to the database since the database was activated, and not for the monitoring interval. However, it provides a useful barometer of the workload none the less, if the representative monitoring interval is valid.

7. Insert **sar**, **vmstat** and **get snapshot for db ..** command information and capture ratio into the FEDWH.FEDWH_INSTANCE table.

The average CPU utilization, memory utilization, maximum concurrent connections, and capture ratio computed for this monitoring interval may be inserted into the FEDWH.FEDWH_INSTANCE table, as shown in Example 5-14.

Example 5-14 Insert sar and vmstat info into FEDWH.FEDWH_INSTANCE table

```

INSERT INTO FEDWH.FEDWH_INSTANCE
(S_SNAPID, E_SNAPID, S_SNAPSHOT_TIMESTAMP, E_SNAPSHOT_TIMESTAMP,
MAXIMUMCONNECTIONS)
SELECT DISTINCT T2.SNAPID,T1.SNAPID, T2.SNAPSHOT_TIMESTAMP,
T1.SNAPSHOT_TIMESTAMP,
(SELECT CONNECTIONS_TOP FROM TABLE(SNAPSHOT_DATABASE(' ', -2)) AS
                                                                    SNAP_DB)
FROM (SELECT * FROM FEDWH.FEDWH_SNAPSHOT_DYN_SQL
WHERE SNAPID= PREVVAL FOR FEDWH.GENSNAPID) T1
INNER JOIN FEDWH.FEDWH_SNAPSHOT_DYN_SQL T2 ON T1.SNAPID-1=T2.SNAPID;

UPDATE FEDWH.FEDWH_INSTANCE SET SAR_CPU=:averagecpu WHERE E_SNAPID=PREVVAL FOR
FEDWH.GENSNAPID;

UPDATE FEDWH.FEDWH_INSTANCE SET MEMORYCONSUMED=:memoryconsumed WHERE
E_SNAPID=PREVVAL FOR FEDWH.GENSNAPID;

UPDATE
(SELECT SAR_CPU, CAPTURE_RATIO, S_SNAPID, E_SNAPID, S_SNAPSHOT_TIMESTAMP,
E_SNAPSHOT_TIMESTAMP FROM FEDWH.FEDWH_INSTANCE
WHERE E_SNAPID=PREVVAL FOR FEDWH.GENSNAPID) as inst
SET CAPTURE_RATIO= (((SELECT DECIMAL(SUM(TOTAL_SYS_CPU_TIME+
TOTAL_USR_CPU_TIME)) from FEDWH.FEDWH_SNAPSHOT_DYN_SQL_INTERVAL dynsql WHERE
dynsql.S_SNAPID=inst.S_SNAPID and dynsql.E_SNAPID=inst.E_SNAPID)
/8)/TIMESTAMPDIFF(2,CHAR(inst.E_SNAPSHOT_TIMESTAMP-inst.S_SNAPSHOT_TIMESTAMP))*
100)/SAR_CPU;

```

In Example 5-14, the highlighted value '8' is the number of processors in our system. The capture ratio represents the discrepancy in the CPU utilization as captured in the dynamic cache from all the queries versus the utilization

recorded by the operating system **sar** command during the monitoring interval.

The capture ratio is the ratio of the CPU utilization derived from the cumulative cost for all queries in the dynamic cache with the CPU utilization obtained from the **sar** command (SAR_CPU column in the FEDWH.FEDWH_INSTANCE table for the particular monitoring interval) within a given monitoring interval, as follows:

$$\text{CPU utilization as computed from the dynamic cache metrics (CPU DC)} = \frac{((\text{Sum of all queries' (TOTAL_USR_CPU_TIME + TOTAL_SYS_CPU_TIME)}) / (\text{number of CPUs}))}{(\text{monitoring interval})}$$
$$\text{Capture ratio (CR)} = (\text{CPU DC} / \text{SAR_CPU})$$

8. Populate the EXPLAIN tables with SQL statements in the dynamic cache.

The objective is to determine the access paths of the dynamic SQL statements in the cache and identify the remote SQL fragments corresponding to each user query.

Example 5-15 shows the script that extracts the user SQL (ignores the remote SQL fragments by filtering out rows where the CPU time is zero), and then populates the EXPLAIN tables with the application user's schema.

Example 5-15 Populate the EXPLAIN tables

```
db2 connect to fedserv user fedwh using fedwh
# Write the extracted contents of the dynamic cache to the checkstatements.out
#file
rm checkstatements.out
db2 -x -r checkstatements.out
"SELECT STMT_TEXT
FROM TABLE(SNAPSHOT_DYN_SQL('-',-2)) as dynsql
where NUM_EXECUTIONS > 0 AND
(TOTAL_USR_CPU_TIME+TOTAL_SYS_CPU_TIME) > 0
order by TOTAL_USR_CPU_TIME+TOTAL_SYS_CPU_TIME DESC"
#
#Edit the contents of the checkstatements.out file and write the output to
#checkstatements.sql
#
sed 's/ *$;/;' checkstatements.out>checkstatements.sql
#
#Populate the EXPLAIN tables with all the SQL statements in the
#checkstatements.sql file with the application user's schema
#
db2 "set current explain mode explain"
db2 "set current schema kawa"
db2 -tvf checkstatements.sql
```

```
db2 "set current explain mode no"
db2 "set current schema fedwh"
```

Note: As mentioned earlier, there is no direct mechanism to link the remote SQL fragments in the dynamic cache with their corresponding user SQL statement.

This EXPLAIN output will help us to identify such relationships since **db2exfmt** output provides the SQL fragment text in the RMTQTXT field of the SHIP operator (3 and 6), as shown in Example 5-16.

Example 5-16 Sample db2exfmt output with a SHIP operator and RMTQTXT field

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

***** EXPLAIN INSTANCE *****

DB2_VERSION: 08.02.0
SOURCE_NAME: SQLC2E05
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-08-01-19.26.25.607199
EXPLAIN_REQUESTER: FEDWH

Database Context:

Parallelism: None
CPU Speed: 4.723442e-07
Comm Speed: 100
Buffer Pool size: 78000
Sort Heap size: 20000
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 1020

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 201 -----

QUERYNO: 3
 QUERYTAG: CLP
 Statement Type: Select
 Updatable: No
 Deletable: No
 Query Degree: 1

Original Statement:

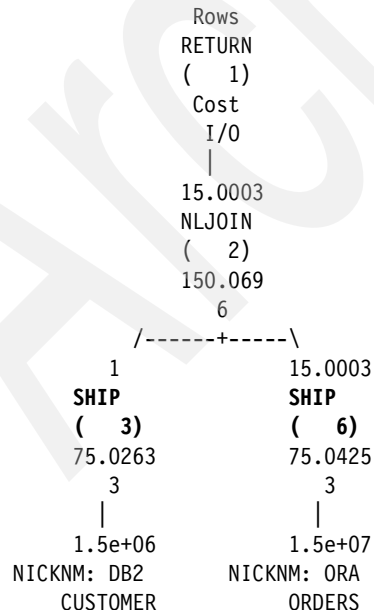
 SELECT 'Q02',C_NAME,O_SHIPPRIORITY
 FROM DB2.CUSTOMER OUTER JOIN ORA.ORDERS on C_CUSTKEY=O_CUSTKEY
 WHERE C_CUSTKEY = 1

Optimized Statement:

 SELECT 'Q02', Q2.C_NAME AS "C_NAME", Q1.O_SHIPPRIORITY AS "O_SHIPPRIORITY"
 FROM ORA.ORDERS AS Q1, DB2.CUSTOMER AS Q2
 WHERE (+0000000001. = Q1.O_CUSTKEY) AND (Q2.C_CUSTKEY = 1)

Access Plan:

 Total Cost: 150.069
 Query Degree: 1



1) RETURN: (Return Result)

Cumulative Total Cost:	150.069
Cumulative CPU Cost:	145648
Cumulative I/O Cost:	6
Cumulative Re-Total Cost:	0.0205163
Cumulative Re-CPU Cost:	43435.1
Cumulative Re-I/O Cost:	0
Cumulative First Row Cost:	150.05
Estimated Bufferpool Buffers:	6.0461
Remote communication cost:	25.7189

Arguments:

BLDLEVEL: (Build level)

DB2 v8.1.1.64 : s040509

ENVVAR : (Environment Variable)

DB2_EXTENDED_OPTIMIZATION = ON

STMTHEAP: (Statement heap size)

8192

Input Streams:

5) From Operator #2

Estimated number of rows:	15.0003
Number of columns:	3
Subquery predicate ID:	Not Applicable

Column Names:

+Q3.0_SHIPRIORITY+Q3.C_NAME+Q3.\$CO

2) NLJOIN: (Nested Loop Join)

Cumulative Total Cost:	150.069
Cumulative CPU Cost:	145648
Cumulative I/O Cost:	6
Cumulative Re-Total Cost:	0.0205163
Cumulative Re-CPU Cost:	43435.1
Cumulative Re-I/O Cost:	0
Cumulative First Row Cost:	150.05
Estimated Bufferpool Buffers:	6.0461
Remote communication cost:	25.7189

Arguments:

EARLYOUT: (Early Out flag)
 NONE
FETCHMAX: (Override for FETCH MAXPAGES)
 IGNORE
ISCANMAX: (Override for ISCAN MAXPAGES)
 IGNORE

Predicates:

2) Predicate used in Join
 Relational Operator: Equal (=)
 Subquery Input Required: No
 Filter Factor: 1

Predicate Text:

(Q1.0_CUSTKEY = Q2.C_CUSTKEY)

Input Streams:

2) From Operator #3

Estimated number of rows: 1
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.C_NAME+Q2.C_CUSTKEY

4) From Operator #6

Estimated number of rows: 15.0003
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q1.0_SHIPPRIORITY

Output Streams:

5) To Operator #1

Estimated number of rows: 15.0003
Number of columns: 3

Subquery predicate ID: Not Applicable

Column Names:

+Q3.0_SHIPRIORITY+Q3.C_NAME+Q3.\$CO

3) SHIP : (Ship)

Cumulative Total Cost: 75.0263
Cumulative CPU Cost: 55683
Cumulative I/O Cost: 3
Cumulative Re-Total Cost: 0.00726371
Cumulative Re-CPU Cost: 15378
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 75.0251
Estimated Bufferpool Buffers: 3.0461
Remote communication cost: 9.35938

Arguments:

CSERQY : (Remote common subexpression)
FALSE

DSTSEVER: (Destination (ship to) server)
- (NULL).

JN INPUT: (Join input leg)
OUTER

RMTQTXT : (Remote statement)

SELECT A0."C_CUSTKEY", A0."C_NAME" FROM
"TPCD"."CUSTOMER" A0 WHERE (A0."C_CUSTKEY" = 1) FOR READ ONLY

SRCSEVER: (Source (ship from) server)
DB2SERV

STREAM : (Remote stream)
FALSE

Input Streams:

1) From Object DB2.CUSTOMER

Estimated number of rows: 1.5e+06
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.C_NAME

Output Streams:

2) To Operator #2

Estimated number of rows: 1
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.C_NAME+Q2.C_CUSTKEY

6) SHIP : (Ship)

Cumulative Total Cost: 75.0425
Cumulative CPU Cost: 89964.6
Cumulative I/O Cost: 3
Cumulative Re-Total Cost: 25.0149
Cumulative Re-CPU Cost: 31457.1
Cumulative Re-I/O Cost: 1
Cumulative First Row Cost: 75.0252
Estimated Bufferpool Buffers: 4
Remote communication cost: 16.3595

Arguments:

CSERQY : (Remote common subexpression)
FALSE
DSTSEVER: (Destination (ship to) server)
- (NULL).
JN INPUT: (Join input leg)
INNER
RMTQTX : (Remote statement)
SELECT A0."O_SHIPPRIORITY" FROM "IITEST"."ORDERS" A0
WHERE (0000000001. = A0."O_CUSTKEY") AND (A0."O_CUSTKEY" = :H0)
SRCSEVER: (Source (ship from) server)
ORASERV
STREAM : (Remote stream)
FALSE

Input Streams:

3) From Object ORA.ORDERS

Estimated number of rows: 1.5e+07
Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:

+Q1.\$RID\$+Q1.O_SHIPPRIORITY+Q1.O_CUSTKEY

Output Streams:

4) To Operator #2

Estimated number of rows: 15.0003
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q1.0_SHIPPRIORITY

Objects Used in Access Plan:

Schema: DB2
Name: CUSTOMER
Type: Nickname

Time of creation: 2004-06-11-21.33.19.191172
Last statistics update: 2004-06-11-22.37.58.434937
Number of columns: 8
Number of rows: 1500000
Width of rows: 41
Number of buffer pool pages: 69156
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: ORA
Name: ORDERS
Type: Nickname

Time of creation: 2004-06-11-21.33.10.349783
Last statistics update: 2004-06-11-22.32.45.545670
Number of columns: 9
Number of rows: 15000000
Width of rows: 38
Number of buffer pool pages: 443840
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node

```
Prefetch page count:          32
Container extent page count:   32
Executing Connect Reset -- Connect Reset was Successful.
```

Attention: This EXPLAIN output can also be used to monitor changes in access paths for a given SQL query over time, and should be used to select the proper reporting interval as discussed in “Step 4: Identify reporting interval” on page 414.

9. Link the EXPLAIN output to the particular monitoring interval.

The EXPLAIN output is associated with a particular monitoring interval, and this relationship must be recorded in the FEDWH.FEDWH_EXPLAIN_INSTANCE table, as shown in Example 5-17, using the SNAPID value of the end of the monitoring interval as the identifier.

Example 5-17 Link EXPLAIN output to particular monitoring interval using SNAPID

```
INSERT INTO FEDWH.FEDWH_EXPLAIN_INSTANCE
SELECT PREVVAL FOR FEDWH.GENSNAPID, T2.EXPLAIN_REQUESTER,T2.EXPLAIN_TIME,
       T2.SOURCE_NAME,T2.SOURCE_SCHEMA,T2.SOURCE_VERSION
FROM FEDWH.EXPLAIN_INSTANCE AS T2
LEFT OUTER JOIN FEDWH.FEDWH_EXPLAIN_INSTANCE AS T3 ON
(T2.EXPLAIN_REQUESTER,T2.EXPLAIN_TIME,T2.SOURCE_NAME,T2.SOURCE_SCHEMA,T2.SOURCE_VERSION)=(T3.EXPLAIN_REQUESTER,T3.EXPLAIN_TIME,T3.SOURCE_NAME,T3.SOURCE_SCHEMA,T3.SOURCE_VERSION)
WHERE T3.SNAPID IS NULL
;
```

10. Compute the metrics from the dynamic cache for this monitoring interval.

The net statistics for this monitoring interval for each query need to be computed and stored in the FEDWH.FEDWH_SNAPSHOT_DYN_INTERVAL table, as shown in Example 5-18.

Example 5-18 Compute metrics for the monitoring interval and store

```
INSERT INTO FEDWH.FEDWH_SNAPSHOT_DYN_SQL_INTERVAL
SELECT T3.S_SNAPID, T3.E_SNAPID, T3.S_SNAPSHOT_TIMESTAMP,
       T3.E_SNAPSHOT_TIMESTAMP,
       T1.ROWS_READ-COALESCE(T2.ROWS_READ,0) AS ROWS_READ,
       T1.ROWS_WRITTEN-COALESCE(T2.ROWS_WRITTEN,0) AS ROWS_WRITTEN,
       T1.NUM_EXECUTIONS-COALESCE(T2.NUM_EXECUTIONS,0) AS NUM_EXECUTIONS,
       T1.NUM_COMPILATIONS-COALESCE(T2.NUM_COMPILATIONS,0) AS NUM_COMPILATIONS,
       T1.PREP_TIME_WORST, T1.PREP_TIME_BEST,
       T1.INT_ROWS_DELETED-COALESCE(T2.INT_ROWS_DELETED,0) AS INT_ROWS_DELETED,
       T1.INT_ROWS_INSERTED-COALESCE(T2.INT_ROWS_INSERTED,0) AS
           INT_ROWS_INSERTED,
       T1.INT_ROWS_UPDATED-COALESCE(T2.INT_ROWS_UPDATED,0) AS INT_ROWS_UPDATED,
```

```

T1.STMT_SORTS-COALESCE(T2.STMT_SORTS,0) AS STMT_SORTS,
T1.TOTAL_EXEC_TIME-COALESCE(T2.TOTAL_EXEC_TIME,0) AS TOTAL_EXEC_TIME,
T1.TOTAL_SYS_CPU_TIME-COALESCE(T2.TOTAL_SYS_CPU_TIME,0) AS
TOTAL_SYS_CPU_TIME,
T1.TOTAL_USR_CPU_TIME-COALESCE(T2.TOTAL_USR_CPU_TIME,0) AS
TOTAL_USR_CPU_TIME, T1.STMT_TEXT
FROM FEDWH.FEDWH_INSTANCE T3
INNER JOIN FEDWH.FEDWH_SNAPSHOT_DYN_SQL T1 ON T1.SNAPID=T3.E_SNAPID
LEFT OUTER JOIN FEDWH.FEDWH_SNAPSHOT_DYN_SQL T2 ON
VARCHAR(T1.STMT_TEXT)=VARCHAR(T2.STMT_TEXT) AND T2.SNAPID=T3.S_SNAPID
WHERE T1.NUM_EXECUTIONS-COALESCE(T2.NUM_EXECUTIONS,0)>0
AND T3.E_SNAPID=PREVVAL FOR FEDWH.GENSNAPID;

```

Important: The statement text of the user-entered query and the query fragment text are both stored as CLOBs (which can be up to 2 gigabytes long). In Example 5-18 we had to cast the STMT_TEXT column as VARCHAR (maximum of 32672 bytes long) in order to perform a comparison. Should the statement text exceed 32672 bytes, the cast function truncates the data returned from the CLOB to 32672 bytes and a false match may occur. A warning message such as “SQL0445W Value “SELECT A0.”L_ORDERKEY”, A0.”L_EXTENDEDPRICE”, A0.”L_DISCOUNT” has been truncated.” is issued.

Figure 5-5 on page 407 through Figure 5-8 on page 410 show the contents of the FEDWH.FEDWH_SNAPSHOT_DYN_INTERVAL table. It has information relating to many intervals.

Command Editor 1						
Command Editor Selected Edit View Tools Help						
Commands Query Results Access Plan						
S_SNAPID	E_SNAPID	S_SNAPSHOT_TIMESTAMP	E_SNAPSHOT_TIMESTAMP	ROWS_READ	ROW	
1	2	2004/08/01 19:12:19 166037	2004/08/01 19:22:37 448710	0		
1	2	2004/08/01 19:12:19 166037	2004/08/01 19:22:37 448710	0		
1	2	2004/08/01 19:12:19 166037	2004/08/01 19:22:37 448710	0		
1	2	2004/08/01 19:12:19 166037	2004/08/01 19:22:37 448710	0		
1	2	2004/08/01 19:12:19 166037	2004/08/01 19:22:37 448710	0		
1	2	2004/08/01 19:12:19 166037	2004/08/01 19:22:37 448710	0		
1	2	2004/08/01 19:12:19 166037	2004/08/01 19:22:37 448710	0		
1	2	2004/08/01 19:12:19 166037	2004/08/01 19:22:37 448710	1196754		
1	2	2004/08/01 19:12:19 166037	2004/08/01 19:22:37 448710	437910		
1	2	2004/08/01 19:12:19 166037	2004/08/01 19:22:37 448710	29375		
1	2	2004/08/01 19:12:19 166037	2004/08/01 19:22:37 448710	4492064		
1	2	2004/08/01 19:12:19 166037	2004/08/01 19:22:37 448710	29315		
1	2	2004/08/01 19:12:19 166037	2004/08/01 19:22:37 448710	29193		
3	4	2004/08/01 20:07:05 716725	2004/08/01 20:17:37 887336	0		
3	4	2004/08/01 20:07:05 716725	2004/08/01 20:17:37 887336	0		
3	4	2004/08/01 20:07:05 716725	2004/08/01 20:17:37 887336	0		
3	4	2004/08/01 20:07:05 716725	2004/08/01 20:17:37 887336	0		
3	4	2004/08/01 20:07:05 716725	2004/08/01 20:17:37 887336	0		
3	4	2004/08/01 20:07:05 716725	2004/08/01 20:17:37 887336	0		
3	4	2004/08/01 20:07:05 716725	2004/08/01 20:17:37 887336	1263286		
3	4	2004/08/01 20:07:05 716725	2004/08/01 20:17:37 887336	465345		
3	4	2004/08/01 20:07:05 716725	2004/08/01 20:17:37 887336	31187		
3	4	2004/08/01 20:07:05 716725	2004/08/01 20:17:37 887336	4742590		
3	4	2004/08/01 20:07:05 716725	2004/08/01 20:17:37 887336	31079		
3	4	2004/08/01 20:07:05 716725	2004/08/01 20:17:37 887336	31023		
5	6	2004/08/01 21:16:44 811463	2004/08/01 21:28:56 064160	0		

Commit Roll Back Fetch More Rows

☐ Automatically commit updates 52 row(s) in memory

Figure 5-5 FEDWH.FEDWH_SNAPSHOT_DYN_INTERVAL table (1 of 4)

Command Editor 1					
Command Editor Selected Edit View Tools Help					
Commands Query Results Access Plan					
WVS_WRITTEN	NUM_EXECUTIONS	NUM_COMPILATIONS	PREP_TIME_WORST	PREP_TIME_BEST	INT_ROWS_DELE
0	29315	0	51	51	
0	29375	0	26	26	
0	9809	0	44	44	
0	1	1	34	34	
12	1	0	35	35	
0	1	0	1	1	
0	29193	0	21	21	
0	4492614	9810	0	0	
0	29194	29194	0	0	
0	29375	29375	0	0	
0	9808	9808	0	0	
0	29315	29315	0	0	
0	29193	29193	0	0	
0	31079	0	51	25	
0	31186	0	37	26	
0	10356	0	154	44	
76	1	0	35	35	
0	1	0	1	1	
0	31023	0	28	21	
0	4742493	10357	0	0	
0	31023	31023	0	0	
0	31187	31187	0	0	
0	10355	10355	0	0	
0	31079	31079	0	0	
0	31023	31023	0	0	
0	30330	0	51	25	

Commit Roll Back Fetch More Rows

☐ Automatically commit updates 52 row(s) in memory

Figure 5-6 FEDWH.FEDWH_SNAPSHOT_DYN_INTERVAL table (2 of 4)

Command Editor 1			
Command Editor Selected Edit View Tools Help			
Commands Query Results Access Plan			
TOTAL_SYS_CPU_TIME	TOTAL_USR_CPU_TIME	STMT_TEXT	
1	10	SELECT 'Q00',C_NAME FROM DB2.CUSTOMER WHERE C_CUSTKEY ...	
5	18	SELECT 'Q01',C_NAME FROM MSS.CUSTOMER WHERE C_CUSTKEY ...	
114	553	SELECT 'Q03_q3', L_ORDERKEY, SUM(L_EXTENDEDPRI*(1-...	
0	0	SELECT COUNT(*) FROM SYSCAT.PROCEDURES WHERE PROCNAME=...	
0	0	INSERT INTO FEDWH.FEDWH_SNAPSHOT_DYN_SQL SELECT PREVVA...	
0	0	VALUES(NEXTVAL FOR FEDWH.GENSNAPID)	
2	21	SELECT 'Q02',C_NAME,O_SHIPPRIORITY FROM DB2.CUSTOMER OUTE...	
0	0	[ORASERV] SELECT A0."L_ORDERKEY", A0."L_EXTENDEDPRI", A0....	
0	0	[ORASERV] SELECT A0."O_SHIPPRIORITY" FROM "IITEST"."ORDERS" ...	
0	0	[SQLSERV] SELECT A0."C_NAME" FROM "TPCD"."CUSTOMER" A0 WH...	
0	0	[DB2SERV] SELECT A1."O_ORDERKEY", A1."O_ORDERDATE", A1."O_...	
0	0	[DB2SERV] SELECT A0."C_NAME" FROM "TPCD"."CUSTOMER" A0 WH...	
0	0	[DB2SERV] SELECT A0."C_CUSTKEY", A0."C_NAME" FROM "TPCD"."C...	
2	10	SELECT 'Q00',C_NAME FROM DB2.CUSTOMER WHERE C_CUSTKEY ...	
5	20	SELECT 'Q01',C_NAME FROM MSS.CUSTOMER WHERE C_CUSTKEY ...	
118	590	SELECT 'Q03_q3', L_ORDERKEY, SUM(L_EXTENDEDPRI*(1-...	
0	0	INSERT INTO FEDWH.FEDWH_SNAPSHOT_DYN_SQL SELECT PREVVA...	
0	0	VALUES(NEXTVAL FOR FEDWH.GENSNAPID)	
3	21	SELECT 'Q02',C_NAME,O_SHIPPRIORITY FROM DB2.CUSTOMER OUTE...	
0	0	[ORASERV] SELECT A0."L_ORDERKEY", A0."L_EXTENDEDPRI", A0....	
0	0	[ORASERV] SELECT A0."O_SHIPPRIORITY" FROM "IITEST"."ORDERS" ...	
0	0	[SQLSERV] SELECT A0."C_NAME" FROM "TPCD"."CUSTOMER" A0 WH...	
0	0	[DB2SERV] SELECT A1."O_ORDERKEY", A1."O_ORDERDATE", A1."O_...	
0	0	[DB2SERV] SELECT A0."C_NAME" FROM "TPCD"."CUSTOMER" A0 WH...	
0	0	[DB2SERV] SELECT A0."C_CUSTKEY", A0."C_NAME" FROM "TPCD"."C...	
2	11	SELECT 'Q00',C_NAME FROM DB2.CUSTOMER WHERE C_CUSTKEY ...	

Figure 5-8 FEDWH.FEDWH_SNAPSHOT_DYN_INTERVAL table (4 of 4)

Step 3: Summarize monitored intervals information

The information needed for capacity planning resides in the operating system command output, dynamic cache (FEDWH.FEDWH_SNAPSHOT_DYN_SQL table), and the explain tables. In this step we combine the information in the dynamic cache and explain tables and record them in the FEDWH.FEDWH_FEDSQL_INTERVAL table, as shown in Example 5-19 on page 411. Figure 5-9 on page 413 and Figure 5-10 on page 414 show the contents of this table.

The FEDWH.FEDWH_FEDSQL_INTERVAL table correlates the user-entered query, the fragments associated with it (using the explain table contents), and computes per-query metrics such as the average CPU time.

Example 5-19 Summarize monitored intervals

```
INSERT INTO FEDWH.FEDWH_FEDSQL_INTERVAL
WITH INST AS (
SELECT
    S_SNAPID
    ,E_SNAPID
    FROM FEDWH.FEDWH_INSTANCE INST
    WHERE E_SNAPID=PREVVAL FOR FEDWH.GENSNAPID
)
,EXPLN AS (
SELECT
    ESTMT2.STATEMENT_TEXT
    ,EARG1.RMTQTXT
    ,EARG1.SRCSEVER
    FROM INST
    INNER JOIN FEDWH.FEDWH_EXPLAIN_INSTANCE EINST ON INST.E_SNAPID=EINST.SNAPID
    INNER JOIN FEDWH.EXPLAIN_STATEMENT ESTMT ON
    EINST.EXPLAIN_REQUESTER=ESTMT.EXPLAIN_REQUESTER AND
    EINST.EXPLAIN_TIME=ESTMT.EXPLAIN_TIME AND EINST.SOURCE_NAME=ESTMT.SOURCE_NAME
    AND EINST.SOURCE_SCHEMA=ESTMT.SOURCE_SCHEMA AND
    EINST.SOURCE_VERSION=ESTMT.SOURCE_VERSION
    --TO GET ,STATEMENT_TEXT
    INNER JOIN (
        SELECT ESTMT2.EXPLAIN_REQUESTER, ESTMT2.EXPLAIN_TIME, ESTMT2.SOURCE_NAME,
        ESTMT2.SOURCE_SCHEMA, ESTMT2.SOURCE_VERSION, ESTMT2.EXPLAIN_LEVEL,
        ESTMT2.STMTNO, ESTMT2.SECTNO
        ,STATEMENT_TEXT
        FROM FEDWH.EXPLAIN_STATEMENT ESTMT2
        WHERE
            ESTMT2.EXPLAIN_LEVEL='0'
    ) ESTMT2 ON ESTMT2.EXPLAIN_REQUESTER=ESTMT.EXPLAIN_REQUESTER AND
    ESTMT2.EXPLAIN_TIME=ESTMT.EXPLAIN_TIME AND ESTMT2.SOURCE_NAME=ESTMT.SOURCE_NAME
    AND ESTMT2.SOURCE_SCHEMA=ESTMT.SOURCE_SCHEMA AND
    ESTMT2.SOURCE_VERSION=ESTMT.SOURCE_VERSION AND ESTMT2.STMTNO=ESTMT.STMTNO AND
    ESTMT2.SECTNO=ESTMT.SECTNO
    --TO GET RMTQTXT AND SRCSEVER
    LEFT OUTER JOIN (
        SELECT EARG1.EXPLAIN_REQUESTER, EARG1.EXPLAIN_TIME, EARG1.SOURCE_NAME,
        EARG1.SOURCE_SCHEMA, EARG1.SOURCE_VERSION, EARG1.EXPLAIN_LEVEL, EARG1.STMTNO,
        EARG1.SECTNO
        ,COALESCE(EARG1.ARGUMENT_VALUE,EARG1.LONG_ARGUMENT_VALUE) AS RMTQTXT
        ,CAST(EARG2.ARGUMENT_VALUE AS VARCHAR(8)) AS SRCSEVER
        FROM FEDWH.EXPLAIN_ARGUMENT EARG1
        INNER JOIN FEDWH.EXPLAIN_ARGUMENT EARG2 ON
        EARG2.EXPLAIN_REQUESTER=EARG1.EXPLAIN_REQUESTER AND
        EARG2.EXPLAIN_TIME=EARG1.EXPLAIN_TIME AND EARG2.SOURCE_NAME=EARG1.SOURCE_NAME
        AND EARG2.SOURCE_SCHEMA=EARG1.SOURCE_SCHEMA AND
```

```

EARG2.SOURCE_VERSION=EARG1.SOURCE_VERSION AND EARG2.STMTNO=EARG1.STMTNO AND
EARG2.SECTNO=EARG1.SECTNO
      AND EARG1.OPERATOR_ID=EARG2.OPERATOR_ID
WHERE
      EARG1.ARGUMENT_TYPE='RMTQTX'
      AND EARG1.EXPLAIN_LEVEL='P'
      AND EARG2.ARGUMENT_TYPE='SRCSEVER'
      AND EARG2.EXPLAIN_LEVEL='P'
) EARG1 ON EARG1.EXPLAIN_REQUESTER=ESTMT.EXPLAIN_REQUESTER AND
EARG1.EXPLAIN_TIME=ESTMT.EXPLAIN_TIME AND EARG1.SOURCE_NAME=ESTMT.SOURCE_NAME
AND EARG1.SOURCE_SCHEMA=ESTMT.SOURCE_SCHEMA AND
EARG1.SOURCE_VERSION=ESTMT.SOURCE_VERSION AND EARG1.STMTNO=ESTMT.STMTNO AND
EARG1.SECTNO=ESTMT.SECTNO
  WHERE ESTMT.EXPLAIN_LEVEL='O'
)
,DYNSQL AS
(
SELECT
      INST.S_SNAPID
      ,INST.E_SNAPID
      ,STMT_TEXT
      ,NUM_EXECUTIONS
      ,DECIMAL(TOTAL_USR_CPU_TIME+TOTAL_SYS_CPU_TIME)/DECIMAL(NUM_EXECUTIONS)
AS AVG_CPU_TIME
      ,CAST(DECIMAL(ROWS_READ)/DECIMAL(NUM_EXECUTIONS) AS DECIMAL(31,2)) AS
AVG_ROWS_READ
      ,CAST(DECIMAL(STMT_SORTS)/DECIMAL(NUM_EXECUTIONS) AS DECIMAL(31,1)) AS
AVG_STMT_SORTS
      FROM INST
      INNER JOIN FEDWH.FEDWH_SNAPSHOT_DYN_SQL_INTERVAL DYNSQL ON
INST.S_SNAPID=DYNSQL.S_SNAPID AND INST.E_SNAPID=DYNSQL.E_SNAPID
)
SELECT
      PDYNSQL.S_SNAPID
      ,PDYNSQL.E_SNAPID
      ,PDYNSQL.STMT_TEXT AS STATEMENT_TEXT
      ,PDYNSQL.NUM_EXECUTIONS
      ,PDYNSQL.AVG_CPU_TIME
      ,PDYNSQL.AVG_STMT_SORTS
      ,EXPLN.SRCSEVER
      ,CDYNSQL.STMT_TEXT AS RMTQTX
      ,CAST(DECIMAL(CDYNSQL.NUM_EXECUTIONS)/DECIMAL(PDYNSQL.NUM_EXECUTIONS)AS
DECIMAL(31,1)) AS RM_AVG_NUM_EXEC
      ,CDYNSQL.AVG_ROWS_READ AS RM_AVG_ROWS
      FROM DYNSQL AS PDYNSQL
      LEFT OUTER JOIN EXPLN ON
VARCHAR(PDYNSQL.STMT_TEXT)=VARCHAR(EXPLN.STATEMENT_TEXT)
      LEFT OUTER JOIN DYNSQL AS CDYNSQL ON
VARCHAR(CDYNSQL.STMT_TEXT)=VARCHAR('['||EXPLN.SRCSEVER||'] '||EXPLN.RMTQTX)

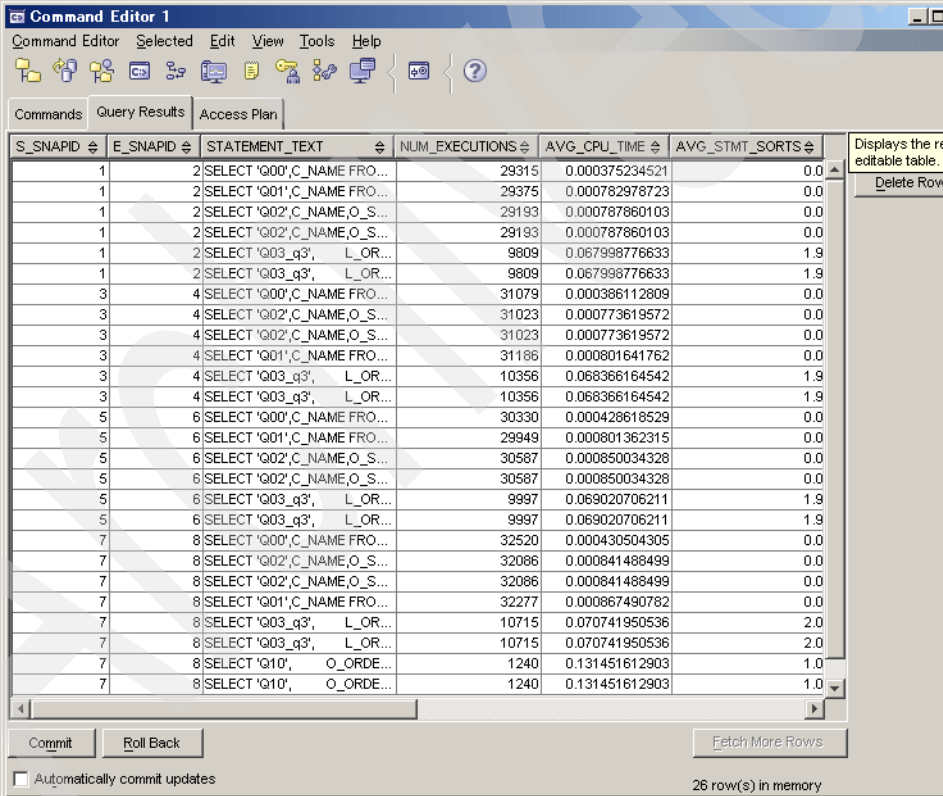
```

```

WHERE PDYNSQL.AVG_CPU_TIME>0
ORDER BY PDYNSQL.AVG_CPU_TIME
;

```

Important: The statement text of the user-entered query and the query fragment text are both stored as CLOBs (which can be up to 2 gigabytes long). In Example 5-19 we had to cast the STMT_TEXT column as VARCHAR (maximum of 32672 bytes long) in order to perform a comparison. Should the statement text exceed 32672 bytes, the cast function truncates the data returned from the CLOB to 32672 bytes and a false match may occur. A warning message “SQL0445W Value "SELECT A0."L_ORDERKEY", A0."L_EXTENDEDPRICE", A0."L_DISCOUNT" has been truncated.” is issued.



S_SNAPID	E_SNAPID	STATEMENT_TEXT	NUM_EXECUTIONS	AVG_CPU_TIME	AVG_STMT_SORTS
1	2	SELECT 'Q00',C_NAME FRO...	29315	0.000375234521	0.0
1	2	SELECT 'Q01',C_NAME FRO...	29375	0.000782978723	0.0
1	2	SELECT 'Q02',C_NAME,O_S...	29193	0.000787860103	0.0
1	2	SELECT 'Q02',C_NAME,O_S...	29193	0.000787860103	0.0
1	2	SELECT 'Q03_g3', L_OR...	9809	0.067998776633	1.9
1	2	SELECT 'Q03_g3', L_OR...	9809	0.067998776633	1.9
3	4	SELECT 'Q00',C_NAME FRO...	31079	0.000386112809	0.0
3	4	SELECT 'Q02',C_NAME,O_S...	31023	0.000773619572	0.0
3	4	SELECT 'Q02',C_NAME,O_S...	31023	0.000773619572	0.0
3	4	SELECT 'Q01',C_NAME FRO...	31186	0.000801641762	0.0
3	4	SELECT 'Q03_g3', L_OR...	10356	0.068366164542	1.9
3	4	SELECT 'Q03_g3', L_OR...	10356	0.068366164542	1.9
5	6	SELECT 'Q00',C_NAME FRO...	30330	0.000428618529	0.0
5	6	SELECT 'Q01',C_NAME FRO...	29949	0.000801362315	0.0
5	6	SELECT 'Q02',C_NAME,O_S...	30587	0.000850034328	0.0
5	6	SELECT 'Q02',C_NAME,O_S...	30587	0.000850034328	0.0
5	6	SELECT 'Q03_g3', L_OR...	9997	0.069020706211	1.9
5	6	SELECT 'Q03_g3', L_OR...	9997	0.069020706211	1.9
7	8	SELECT 'Q00',C_NAME FRO...	32520	0.000430504305	0.0
7	8	SELECT 'Q02',C_NAME,O_S...	32086	0.000841488499	0.0
7	8	SELECT 'Q02',C_NAME,O_S...	32086	0.000841488499	0.0
7	8	SELECT 'Q01',C_NAME FRO...	32277	0.000867490782	0.0
7	8	SELECT 'Q03_g3', L_OR...	10715	0.070741950536	2.0
7	8	SELECT 'Q03_g3', L_OR...	10715	0.070741950536	2.0
7	8	SELECT 'Q10', O_ORDE...	1240	0.131451612903	1.0
7	8	SELECT 'Q10', O_ORDE...	1240	0.131451612903	1.0

Figure 5-9 Contents of FEDWH.FEDWH_FEDSQL_INTERVAL (1 of 2)

SRCSEVER	RMTQTX	RM_AVG_NUM_EXEC	RM_AVG_ROWS
DB2SERV	[DB2SERV] SELECT A0."C_NAME" FROM "TPCD"."CUSTOMER" ...	1.0	1.00
SQLSERV	[SQLSERV] SELECT A0."C_NAME" FROM "TPCD"."CUSTOMER" ...	1.0	1.00
DB2SERV	[DB2SERV] SELECT A0."C_CUSTKEY", A0."C_NAME" FROM "T...	1.0	1.00
ORASERV	[ORASERV] SELECT A0."O_SHIPPRIORITY" FROM "TITEST"."OR...	1.0	15.00
ORASERV	[ORASERV] SELECT A0."L_ORDERKEY", A0."L_EXTENDEDPR...	458.0	0.26
DB2SERV	[DB2SERV] SELECT A1."O_ORDERKEY", A1."O_ORDERDATE", ...	0.9	458.00
DB2SERV	[DB2SERV] SELECT A0."C_NAME" FROM "TPCD"."CUSTOMER" ...	1.0	1.00
DB2SERV	[DB2SERV] SELECT A0."C_CUSTKEY", A0."C_NAME" FROM "T...	1.0	1.00
ORASERV	[ORASERV] SELECT A0."O_SHIPPRIORITY" FROM "TITEST"."OR...	1.0	15.00
SQLSERV	[SQLSERV] SELECT A0."C_NAME" FROM "TPCD"."CUSTOMER" ...	1.0	1.00
DB2SERV	[DB2SERV] SELECT A1."O_ORDERKEY", A1."O_ORDERDATE", ...	0.9	458.00
ORASERV	[ORASERV] SELECT A0."L_ORDERKEY", A0."L_EXTENDEDPR...	457.9	0.26
DB2SERV	[DB2SERV] SELECT A0."C_NAME" FROM "TPCD"."CUSTOMER" ...	0.9	1.00
SQLSERV	[SQLSERV] SELECT A0."C_NAME" FROM "TPCD"."CUSTOMER" ...	1.0	1.00
ORASERV	[ORASERV] SELECT A0."O_SHIPPRIORITY" FROM "TITEST"."OR...	1.0	15.00
DB2SERV	[DB2SERV] SELECT A0."C_CUSTKEY", A0."C_NAME" FROM "T...	1.0	1.00
DB2SERV	[DB2SERV] SELECT A1."O_ORDERKEY", A1."O_ORDERDATE", ...	0.9	458.00
ORASERV	[ORASERV] SELECT A0."L_ORDERKEY", A0."L_EXTENDEDPR...	457.9	0.26
DB2SERV	[DB2SERV] SELECT A0."C_NAME" FROM "TPCD"."CUSTOMER" ...	1.0	1.00
DB2SERV	[DB2SERV] SELECT A0."C_CUSTKEY", A0."C_NAME" FROM "T...	1.0	1.00
ORASERV	[ORASERV] SELECT A0."O_SHIPPRIORITY" FROM "TITEST"."OR...	1.0	15.00
SQLSERV	[SQLSERV] SELECT A0."C_NAME" FROM "TPCD"."CUSTOMER" ...	1.0	1.00
DB2SERV	[DB2SERV] SELECT A1."O_ORDERKEY", A1."O_ORDERDATE", ...	1.0	458.00
ORASERV	[ORASERV] SELECT A0."L_ORDERKEY", A0."L_EXTENDEDPR...	458.0	0.26
DB2SERV	[DB2SERV] SELECT A0."L_ORDERKEY" FROM "TPCD"."LINEITE...	1.0	15,783.00
ORASERV	[ORASERV] SELECT A0."O_ORDERKEY", A0."O_ORDERPRIOR...	1.0	6,255.00

Figure 5-10 Contents of FEDWH.FEDWH_FEDSQL_INTERVAL (2 of 2)

Step 4: Identify reporting interval

The capturing of metrics needs to occur over an extended period of time in order to generate a good sample from which to estimate our capacity requirements.

Eventually, there will be a number of rows for each query corresponding to a particular monitoring interval (identified by the start and end SNAPID pair) in the FEDWH.FEDWH_FEDSQL_INTERVAL table. There will also be multiple entries for each monitoring interval in the FEDWH.FEDWH_INSTANCE table with information regarding average CPU utilization, memory consumption, maximum number of concurrent connections, and capture ratio.

When a large sample is chosen, there can be wide variations in a query's CPU utilization for reasons such as:

- ▶ System environment (hardware/software) has changed
- ▶ Volume of data accessed has changed
- ▶ Access path of the query has changed

- Workload profile has changed—more concurrent connections have increased contention

It is essential to choose a clean reporting interval (set of monitoring intervals to consider for utilization computation) that filters out monitoring intervals that may have metrics that may be distorted by such events, before computing a utilization report.

Some of these changes can be detected with the information collected such as access path change for a given query from the EXPLAIN output¹, and an increase in the number of maximum concurrent connections from the snapshot output. It may be possible to deduce changes in the volume of data from the EXPLAIN output, but these cannot be definitive. However, detecting the other changes requires some other mechanisms to be implemented.

Attention: As mentioned in the assumptions, this publication does not provide guidelines on filtering out monitoring intervals that may distort utilization computations. It is left to the reader to implement appropriate procedures to ensure that a clean reporting interval of monitoring intervals is used for computing utilizations for the various queries.

The metrics relating to the clean reporting interval may be extracted from the FEDWH.FEDWH_FEDSQL_INTERVAL into a separate table if required.

Step 5: Generate utilization report

Once the reporting interval has been defined, one can compute the various utilizations from the FEDWH.FEDWH_FEDSQL_INTERVAL and FEDWH.FEDWH_INSTANCE tables. The utilization information is recorded in the FEDWH.FEDWH_INSTANCE_REPORT and FEDWH.FEDWH_FEDSQL_REPORT tables.

Attention: The worst case values are assumed for the reporting interval—the lowest capture ratio value, highest value of the maximum number of connections, highest value of memory consumed, and highest **sar** CPU utilization value.

Example 5-20 on page 416 shows the SQL for populating the FEDWH.FEDWH_INSTANCE_REPORT and FEDWH.FEDWH_FEDSQL_REPORT tables from the FEDWH.FEDWH_FEDSQL_INTERVAL and FEDWH.FEDWH_INSTANCE

¹ The contents of the FEDWH.EXPLAIN_INSTANCE table, FEDWH.FEDWH_SNAPSHOT_DYN_SQL and the FEDWH.FEDWH_EXPLAIN_INSTANCE tables can provide this information.

tables, while Figure 5-11 on page 417 through Figure 5-13 on page 419 show the contents of the tables.

Example 5-20 Populating the utilization reports

```

INSERT INTO FEDWH.FEDWH_INSTANCE_REPORT
SELECT CAST(1 AS INTEGER) AS REPORT_ID
      , CURRENT DATE AS REPORT_DATE
      , SUM(TIMESTAMPDIFF(2,CHAR(E_SNAPSHOT_TIMESTAMP-S_SNAPSHOT_TIMESTAMP))) AS
TOTAL_INTERVAL
      , MAX(SAR_CPU) AS SAR_CPU
      , MIN(CAPTURE_RATIO) AS CAPTURE_RATIO
      , MAX(MEMORYCONSUMED) AS MEMORYCONSUMED
      , MAX(MAXIMUMCONNECTIONS) AS MAXIMUMCONNECTIONS
FROM FEDWH.FEDWH_INSTANCE
WHERE S_SNAPID IN (1,3);
--
--NOTE: The 1 and 3 values in CAST(1 AS INTEGER) and S_SNAPID IN (1,3)
--shown above have to be selected by the DBA as the range monitoring interval
--to be used in the computations. It is up to the DBA to make a sound judgement
--about choosing only the representative interval for the utilization reports
--
--The following table may not be populated correctly with SQL since the
--block size of the temporary table space is limited to 32K and SQL query text
--may get truncated with a warning message such as the following
--SQL0445W Value "SELECT A0."L_ORDERKEY", A0."L_EXTENDEDPRICE", A0."L_DISCOUNT"
--has been truncated.  SQLSTATE=01004
--In such cases, it might be appropriate to write a program to avoid truncation
--
INSERT INTO FEDWH.FEDWH_FEDSQL_REPORT
WITH FEDINT AS
(SELECT CAST(1 AS INTEGER) AS REPORT_ID,T1.*
  FROM FEDWH.FEDWH_FEDSQL_INTERVAL as T1
  WHERE S_SNAPID IN (1,3)
)
SELECT
  T1.REPORT_ID
 ,T1.STATEMENT_TEXT
 ,NUM_EXECUTIONS
 ,EXEC_PER_SEC
 ,AVG_CPU_TIME
 ,AVG_STMT_SORTS
 ,SRCSEVER
 ,RMTQTX
 ,RM_AVG_NUM_EXEC
 ,RM_AVG_ROWS
FROM
  (SELECT FEDINT.REPORT_ID
    , CAST(STATEMENT_TEXT AS VARCHAR(2700)) AS STATEMENT_TEXT

```

```

        , SUM(NUM_EXECUTIONS) AS NUM_EXECUTIONS
        , CAST(SUM(NUM_EXECUTIONS)/TOTAL_INTERVAL AS DECIMAL(31,10)) AS
EXEC_PER_SEC
        , MAX(AVG_CPU_TIME) AS AVG_CPU_TIME
        , MAX(AVG_STMT_SORTS) AS AVG_STMT_SORTS
FROM FEDWH.FEDWH_INSTANCE_REPORT FEDINS INNER JOIN FEDINT ON
FEDINT.REPORT_ID= FEDINS.REPORT_ID
GROUP BY FEDINT.REPORT_ID,CAST(STATEMENT_TEXT AS
VARCHAR(2700)),TOTAL_INTERVAL
) T1 INNER JOIN
(SELECT CAST(STATEMENT_TEXT AS VARCHAR(2700)) AS STATEMENT_TEXT
, SRCSEVER
, CAST(RMTQTXTEXT AS VARCHAR(500)) AS RMTQTXTEXT
, AVG(RM_AVG_NUM_EXEC) AS RM_AVG_NUM_EXEC
, AVG(RM_AVG_ROWS) AS RM_AVG_ROWS
FROM FEDINT
GROUP BY CAST(STATEMENT_TEXT AS VARCHAR(2700)),SRCSEVER,CAST(RMTQTXTEXT AS
VARCHAR(500))
) T2
ON T1.STATEMENT_TEXT=T2.STATEMENT_TEXT;

```

Important: The statement text of the user-entered query and the query fragment text are both stored as CLOBs (which can be up to 2 gigabytes long). In Example 5-20, we had to cast the STATEMENT_TEXT and RMTQTXTEXT columns as VARCHAR in order to perform a GROUP BY since the block size of the temporary table space is limited to 32 K. We chose to cast STATEMENT_TEXT as 2700 characters, and the RMTQTXTEXT as 500 characters. Should these lengths be exceeded, the report may have false aggregations.

REPORT_ID	REPORT_DATE	TOTAL_INTERVAL	SAR_CPU	CAPTURE_RATIO	MEMORYCONSUMPTION	MAXIMUMCONNECTIONS
1	2004/08/09	1250	18.0	0.813	1353640	24
2	2004/08/09	612	24.0	0.842	1346872	27

Figure 5-11 Contents of FEDWH.FEDWH_INSTANCE_REPORT

Command Editor 2				
Command Editor Selected Edit Tools Help				
Commands Query Results				
SRCSEV...	RMTQTX	RM_AVG_NUM_EXEC	RM_AVG_ROWS	
0.0 DB2SERV	SELECT A0."C_NAME" FROM "TPCD"."CUSTOMER" A0 WHERE (...)	1.0	1.00	Add Row
0.0 SQLSERV	SELECT A0."C_NAME" FROM "TPCD"."CUSTOMER" A0 WHERE (...)	1.0	1.00	Delete Row
0.0 DB2SERV	SELECT A0."C_CUSTKEY", A0."C_NAME" FROM "TPCD"."CUSTO...	1.0	1.00	
0.0 ORASERV	SELECT A0."O_SHIPPRIORITY" FROM "ITEST"."ORDERS" A0 WH...	1.0	15.00	
.9 DB2SERV	SELECT A1."O_ORDERKEY", A1."O_ORDERDATE", A1."O_SHIPP...	0.9	458.00	
.9 ORASERV	SELECT A0."L_ORDERKEY", A0."L_EXTENDEDPRICE", A0."L_DIS...	457.9	0.26	
0.0 DB2SERV	SELECT A0."C_NAME" FROM "TPCD"."CUSTOMER" A0 WHERE (...)	1.0	1.00	
0.0 SQLSERV	SELECT A0."C_NAME" FROM "TPCD"."CUSTOMER" A0 WHERE (...)	1.0	1.00	
0.0 DB2SERV	SELECT A0."C_CUSTKEY", A0."C_NAME" FROM "TPCD"."CUSTO...	1.0	1.00	
0.0 ORASERV	SELECT A0."O_SHIPPRIORITY" FROM "ITEST"."ORDERS" A0 WH...	1.0	15.00	
.0 DB2SERV	SELECT A1."O_ORDERKEY", A1."O_ORDERDATE", A1."O_SHIPP...	1.0	458.00	
.0 ORASERV	SELECT A0."L_ORDERKEY", A0."L_EXTENDEDPRICE", A0."L_DIS...	458.0	0.26	
.0 DB2SERV	SELECT A0."L_ORDERKEY" FROM "TPCD"."LINEITEM" A0 WHER...	1.0	15,783.00	
.0 ORASERV	SELECT A0."O_ORDERKEY", A0."O_ORDERPRIORITY", A0.ROWI...	1.0	6,255.00	
Commit Roll Back Fetch More Rows				
Automatically commit updates				
14 row(s) in memory				

Figure 5-13 Contents of FEDWH.FEDWH_FEDSQL_REPORT table (2 of 2)

Each query's unit CPU utilization cost

This is computed for each query as follows:

$$((TOTAL_USR_CPU_TIME + TOTAL_SYS_CPU_TIME) / (NUM_EXECUTIONS))$$

Attention: In reality, the NUM_EXECUTIONS value must be compared with the NUM_COMPILATIONS value, which specifies the number of different compilations for a specific SQL statement. The NUM_COMPILATIONS should be 1, otherwise it implies that the NUM_EXECUTIONS relate to different access plans given the multiple compilations that occurred.

This is because some SQL statements issued on different schemas, such as “SELECT * FROM TEST”, will appear to be the same statement in the dynamic cache even though they refer to different access plans. The NUM_COMPILATIONS value must be used in conjunction with NUM_EXECUTIONS to determine whether a bad compilation environment may be skewing the results of dynamic SQL snapshot statistics.

If required, one might validate a query’s utilization using the **db2batch** tool, as shown in Example 5-21, since it measures actual execution times. The assumption is that the query is executed in a real production environment. **db2batch** can also be used to measure utilization for static SQL.

Example 5-21 db2 batch output

```
-----
Statement number: 1

SELECT      O_ORDERPRIORITY,      COUNT(*) AS ORDER_COUNT FROM
ORA.ORDERS WHERE      O_ORDERKEY BETWEEN 1 AND 1000000 AND      EXISTS
(SELECT *      FROM      DB2.LINEITEM      WHERE
L_ORDERKEY = O_ORDERKEY      AND      L_COMMITDATE < L_RECEIPTDATE
) GROUP BY      O_ORDERPRIORITY ORDER BY      O_ORDERPRIORITY

O_ORDERPRIORITY  ORDER_COUNT
-----
1-URGENT          45963
2-HIGH            45962
3-MEDIUM         45344
4-NOT SPECIFIED  45856
5-LOW             46207

Number of rows retrieved is:      5
Number of rows sent to output is: 5

.....lines have been removed.....

*** Statement Details ***
```

```

Node where statement is executing          = 0
Statement type                            = DYNAMIC
Statement operation                        = CLOSE
Section number                            = 1
SQL compiler query cost estimate in timerons = 222926
SQL compiler query cardinality estimate    = 5
Statement sorts                           = 1
Total sort time (ms)                      = 742
Sort overflows                            = 0
DMS Rows read                             = 0
DMS Rows written                           = 0
Internal rows deleted                     = 0
Internal rows updated                     = 0
Internal rows inserted                     = 0
Fetch count                               = 5
Statement operation start time             = Wed Jul 7 11:10:46 2004
Statement operation stop time             = Wed Jul 7 11:10:57 2004
Statement cursor name                      = DYNCUR
Authorization ID for statement precompile  = NULLID
Statement package name                    = TOOL1E00
Total User CPU Time of statement (s)       = 7.050000 seconds
Total System CPU Time of statement (s)     = 0.170000 seconds
Subsections                              = 0
Agents working on statement                = 0
Statement Length                           = 408
Dynamic SQL statement text                 =
SELECT      O_ORDERPRIORITY,              COUNT(*) AS ORDER_COUNT FROM
ORA.ORDERS WHERE      O_ORDERKEY BETWEEN 1 AND 1000000 AND EXISTS
(SELECT *      FROM      DB2.LINEITEM      WHERE
L_ORDERKEY = O_ORDERKEY      AND      L_COMMITDATE < L_RECEIPTDATE
) GROUP BY      O_ORDERPRIORITY ORDER BY      O_ORDERPRIORITY

```

.....lines have been removed.....

*** Tablespace Snapshot ***

```

Tablespace Name                          = TEMP4_TS
Buffer pool data logical reads            = 0
Buffer pool data physical reads           = 0
Asynchronous pool data page reads         = 0
Buffer pool data writes                   = 0

```

.....lines have been removed.....

Summary of Results

=====

Elapsed	Agent CPU	Rows	Rows
---------	-----------	------	------

Statement #	Time (s)	Time (s)	Fetches	Printed
1	10.824	7.240	5	5
Arith. mean	10.824	7.24		
Geom. mean	10.824	7.24		

EXPLAIN TABLES have been populated successfully.

The most important field is **Agent CPU Time(s)** in the **Summary of Results** section, as highlighted. Other fields are drawn from snapshot statistics and may not reflect information related to this query since these statistics include other concurrent activity.

Chart the change in the following values for the reporting interval

The values are:

- ▶ Capture ratio (CAPTURERATIO column in the FEDWH.FEDWH_INSTANCE table).
- ▶ Memory utilization (MEMORYCONSUMED column in the FEDWH.FEDWH_INSTANCE table).
- ▶ Maximum number of concurrent connections (MAXIMUMCONNECTIONS column in the FEDWH.FEDWH_INSTANCE table).
- ▶ (Memory utilization) / (Maximum number of concurrent connections). This is an approximation of memory utilization per connection.
- ▶ Number of executions (NUM_EXECUTIONS in the FEDWH.FEDWH_FEDSQL_INTERVAL table) for *each* query in the dynamic cache.

Figure 5-14 on page 423 and Figure 5-15 on page 424 chart the maximum number of connections and number of executions per query over different monitoring intervals.

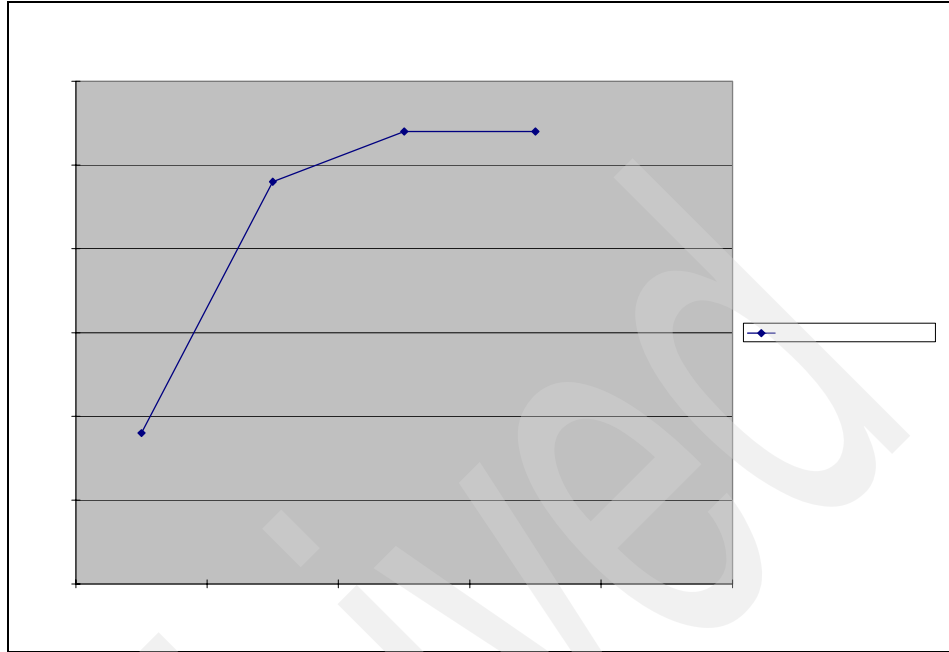


Figure 5-14 Chart of maximum connections per monitoring interval

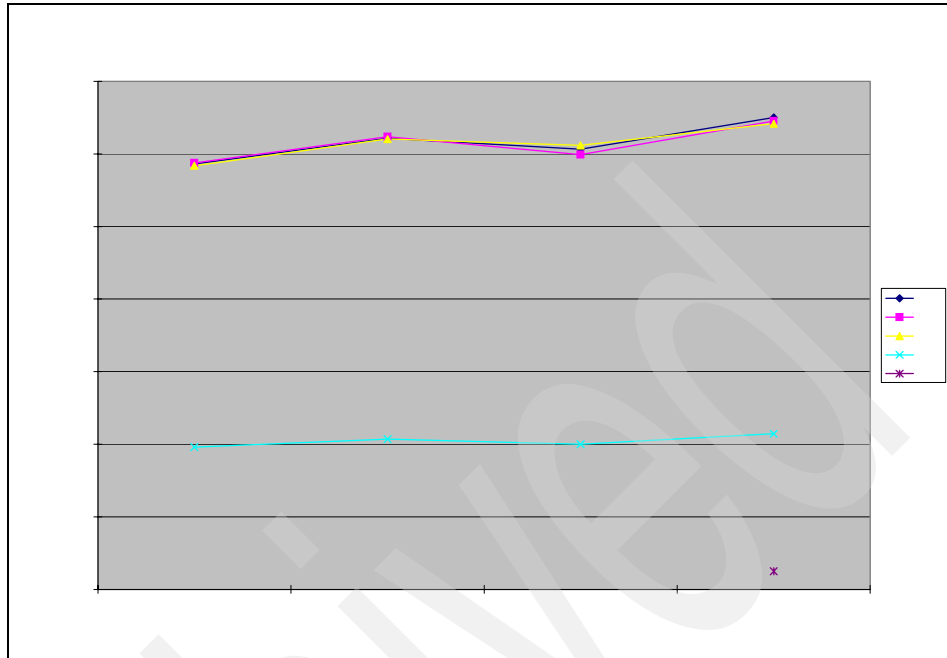


Figure 5-15 Chart of number of executions per query

Note: Charts of these values help determine the average, median, or maximum values in the reporting interval, as well as project future workloads for sizing.

Step 6: Estimate capacity for anticipated future growth

Future growth may involve a combination of one or more of the following occurrences:

- ▶ Introduction of new applications whose query profiles may or may not be known in detail. These are discussed in 5.4, “Capacity planning new applications” on page 427.
- ▶ Change in the profile of existing queries. This may involve changes in the access paths and/or volume of data processed. This could be considered to be new queries whose profiles are better known, and the same approach as discussed in 5.4, “Capacity planning new applications” on page 427, could be applied.
- ▶ Change in the frequency of execution of existing queries where some increase, some reduce, and others disappear altogether.

Important: The capacity planning estimate is most accurate when the profile and frequency of execution is well defined, and the throughput requirement is forecasted adequately.

The charting described in “Step 6: Estimate capacity for anticipated future growth” on page 424 can be used to extrapolate future values for the number of executions for each query, as well as the maximum number of concurrent connections.

Estimate the CPU utilization

The following approach should be used to estimate CPU utilization:

1. Estimate the query workload mix.

If some external domain information is not available, one could project the queries and their frequency of execution using the charting mechanism described.

Compute the concurrent query execution mix by taking the projected maximum number of concurrent connections, and then taking each query and distributing it in proportion to its frequency of execution.

For example, assume the following information has been gathered:

- Query Q1 consumes 0.5 cpu units and executes 100 times in 10 minutes.
- Query Q2 consumes 1.0 cpu units and executes 50 times in 10 minutes.
- Query Q3 consumes 1.5 cpu units and executes 20 times in 10 minutes.
- Projected maximum number of concurrent connections is 50.
- Capture ratio is 90 percent. The dynamic cache utilization only shows 90 percent of what is shown by the `sar` command.

The projected maximum number of concurrent connections of 50 should be assumed to consist of the following concurrent mix of queries Q1, Q2, and Q3:

- Q1 portion of concurrent connections = $50 \times (100 / (100 + 50 + 20)) = 29.4$
- Q2 portion of concurrent connections = $50 \times (50 / (100 + 50 + 20)) = 14.7$
- Q3 portion of concurrent connections = $50 \times (20 / (100 + 50 + 20)) = 5.9$

2. Estimate CPU.

This can be estimated as follows:

$$(29.4 \times 0.5 + 14.7 \times 1 + 5.9 \times 1.5) / 0.9 = 42.5 \text{ CPU units}$$

CPU percentage used of current system = ((CPU estimate) / (number of processors)).

Assuming that the system on which the metrics were collected was a 4-way processor, the CPU percentage would be calculated as follows:

$$42.5 / 4 = 10.625\% \text{ of the current system}$$

If this percent exceeded 80 percent², then a new system should be considered. You need to use the vendor's help in choosing the right system upgrade.

We also need to add a multiprocessing factor at very high utilizations to account for contention—maybe a factor of 10 percent.

Estimate the memory requirements

This is an approximate way of estimating memory requirements. Assume that Table 5-1 shows metrics about maximum concurrent connections, and memory utilization in the reporting interval.

Table 5-1 Memory utilization versus maximum concurrent connections

Concurrent connections	Memory utilization	Memory utilization per connection
15	280 MB	18.7 MB
15	350 MB	23.3 MB
20	400 MB	20 MB
22	500 MB	22.7 MB
19	410 MB	21.6 MB

We can choose average (21.3 MB), median (21.6 MB), or maximum (23.3 MB). Memory utilization should therefore be estimated as:

$$(\text{memory utilization per connection}) \times 50 = 21.6 \times 50 = 1080 \text{ MB}$$

Attention: This approach does not take into account any distortions that may occur due to memory consumption in the file system cache. For the purposes of this exercise, we feel that such distortions can be ignored. However, our intention is to draw attention to this possible distortion, so that the reader may take appropriate action to eliminate this distortion in their sizing exercise (with appropriate tests) if they felt it to be significant in their particular environment.

² Organizations may choose to substitute other values depending upon their individual experiences and practices of their IT environment.

5.4 Capacity planning new applications

The keys to effective capacity planning of new applications are:

- ▶ Having a model of different profiles of queries with their estimated unit cost for a given CPU model.
- ▶ Getting a detailed understanding of the new application's workload, which includes the individual queries and their access characteristics, frequency of execution of each query, and the number of concurrent users.
- ▶ Estimating capacity required for the new application.

Each of these considerations is discussed in the following sections.

5.4.1 Model of different profiles of queries

For organizations that do *not* have an existing DB2 II environment, a model needs to be available from IBM or some other third source that defines categories of queries with unit CPU costs associated with each category.

Organizations with existing DB2 II environments can build a more accurate model of their own using information collected about their existing queries during the various monitoring intervals, as described in “Step 2: Capture runtime metrics” on page 386 and “Step 5: Generate utilization report” on page 415. Besides obtaining superior accuracy of CPU utilizations for a query in one's own environment, this approach provides the DBA with the ability to define one's own custom categories for greater granularity and therefore potentially more accurate capacity estimates for new applications.

Important: The most significant CPU utilization drivers for a federated query are the types of remote data sources such as DB2, Oracle, or SQL Server; the number of rows returned from the remote data sources to the federated server; and the number of interactions between the federated server and the remote data sources.

While other factors, such as the number of rows returned to the user, the maximum number of concurrent users, and the database manager and database configuration parameters (such as SORTHEAP), also impact resource consumption, they tend to have a lesser impact than the type of remote data source and the number of rows returned to the federated server. We therefore recommend that the information identified in Table 5-2 on page 428 be collected for all queries in existing DB2 II environments, and be used as the foundation for defining custom categories for capacity planning of new applications in the existing DB2 II environment.

Table 5-2 Query profile model

Query text	Unit CPU cost	Remote data source type & rows returned	Remote data source type & rows returned

The query text, unit CPU cost, and number of rows from each data source described in Table 5-2 can be obtained from the FEDWH.FEDWH_FEDSQL_REPORT table contents (see Figure 5-12 on page 418 and Figure 5-13 on page 419), while the type of each remote data source (such as Oracle, DB2, or SQL Server) can be determined from the server definition associated with the nickname.

Important: You should also ascertain the maximum number of concurrent connections monitored (from the FEDWH.FEDWH_INSTANCE_REPORT table) in computing the unit costs, and the system configuration involved in these measurements.

5.4.2 Determine new application workload

Identifying the new application’s queries, access characteristics, and workload requires domain expertise that is beyond the scope of this publication.

It is assumed that the DBA has some mechanism of obtaining this information about the new application. It is reasonable to assume that in many cases accurate information will not be forthcoming, and the DBA will have to make an educated guess about the kind of information described in Table 5-2 for each query.

5.4.3 Estimate capacity for the new application

The critical process is choosing corresponding queries in Table 5-2 that most closely resemble each query in the new application.

Once each query in the new application has been categorized using Table 5-2, the unit cost associated with each new query can be used to estimate the capacity requirements of the new application using the same approach described in “Step 6: Estimate capacity for anticipated future growth” on page 424.

DB2 II V8.2 performance enhancements

In this appendix, we provide a high-level overview of the performance enhancements included in DB2 II V8.2.

The topics covered are:

- ▶ Fenced wrappers
- ▶ Parallelism enhancements
- ▶ Updating nickname statistics
- ▶ Cache tables
- ▶ Informational constraints
- ▶ Snapshot monitor enhancements
- ▶ Health Center alerts

Introduction

DB2 II V8.2 has a number of usability, manageability, scalability, and performance enhancements that add value to the business integration environment.

Note: In this appendix, we briefly describe only those performance enhancements in DB2 II Version 8.2 that the application designer and/or DBA need to take specific action to achieve performance benefits.

DB2 II Version 8.2 has a number of performance enhancements that the user automatically benefits from with no specific action required. They include Query Rewrite improvements such as improved UNION ALL processing, and exploitation of intra-partition parallelism in non-partitioned databases on SMP machines. These enhancements are *not* described here.

The following performance enhancements in DB2 II Version 8.2 require the application designer and/or DBA to take specific action to achieve performance benefits:

- ▶ Fenced wrappers
- ▶ Parallelism enhancements
- ▶ Updating nickname statistics
- ▶ Cache tables
- ▶ Informational constraints
- ▶ Snapshot monitor enhancements
- ▶ Health Center alerts

Fenced wrappers

Prior to DB2 II V8.2, wrappers ran in “trusted” mode, which meant that all wrappers ran in the main db2agent created for each user. While this mode facilitates a certain efficiency, it has the following drawbacks:

- ▶ Poorly written wrappers can crash the federated engine.
- ▶ Complicates problem determination.
- ▶ No resource sharing, since each db2agent loads a separate copy of the wrapper and the data source client modules. This has a potential negative impact on scalability due to memory utilization of each db2agent for the wrapper.

In DB2 V8.2, a “fenced” mode wrapper option is provided, which eliminates these disadvantages. A new wrapper option DB2_FENCED is provided, which, when

set to 'Y', indicates fenced mode operation, while a value of 'N' indicates trusted mode operation. The default is trusted mode. The SQL ALTER WRAPPER statement can be used to modify the setting of an existing wrapper.

The advantages of a fenced wrapper are:

- ▶ Allows isolation of wrappers, which protects the federated engine from errant wrapper code, and eases problem determination since it runs in a separate process.
- ▶ Allows resource sharing and improves scalability, since a single copy of the wrapper can be used by multiple db2agent processes. Memory utilization is reduced.
- ▶ Facilitates inter-partition parallelism for nickname data.

Figure A-1 shows the wrapper architecture with trusted and fenced mode.

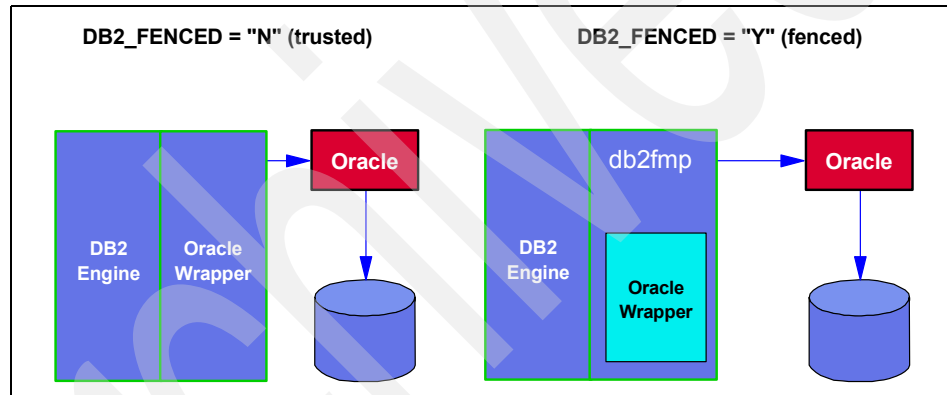


Figure A-1 Wrapper architecture - Fenced and trusted

When the wrapper is first required for a connection for a type of data source (such as when the first user selects from a nickname for a table at a data source of a type supported by the wrapper), DB2 II creates a fenced mode procedure (fmp) process and loads the wrapper into that process. The wrapper, in turn, loads the data source client software modules into the fmp process and makes the connection to the data source. The fmp process can communicate with the user db2agent process. In a partitioned environment, the fmp process can also communicate with db2agntp processes created in each of the partitions during the execution of a query that distributes data from remote data sources to the partitions so it can be joined in parallel with the local data.

If other DB2 II users need data from the same type of data source, and the wrapper and data source client are multi-threaded (such as Oracle, SQL Server, Informix, and DB2), DB2 II uses the same fmp process and establishes another

thread in it to support the connection to the type of data source for the new user. For data source clients that are not thread-safe (such as Sybase and Teradata), DB2 II has to start a new fmp process to load the wrapper and data source client to make connections to different data sources for each user.

Parallelism enhancements

DB2 II Version 8.2 supports both inter-partition and intra-partition parallelism for federated queries.

In the following subsections, we briefly describe the parallelism support:

- ▶ Intra-partition parallelism in a non-DPF environment
- ▶ Inter-partition parallelism in a DPF environment with local data
- ▶ Inter-partition parallelism in a DPF environment without local data

Intra-partition parallelism in a non-DPF environment

Figure A-2 shows intra-partition parallelism support before and after DB2 II V8.2.

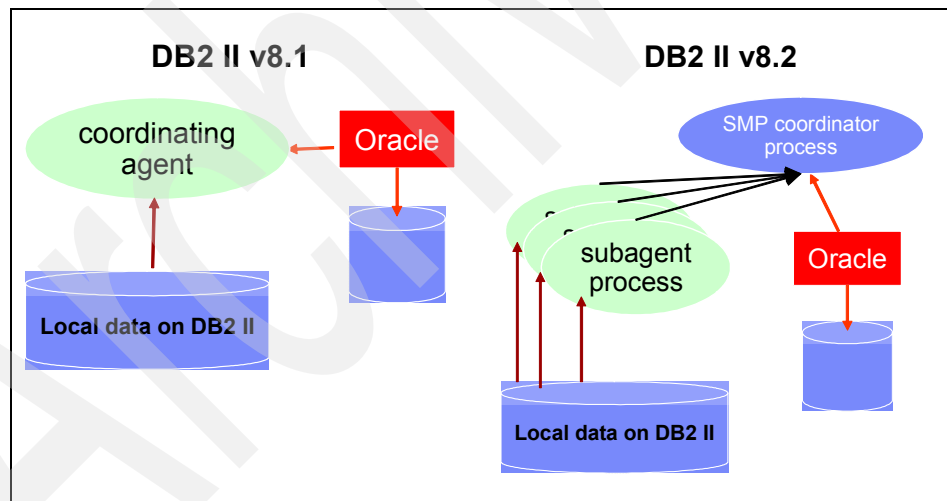


Figure A-2 Intra-partition parallelism on SMP systems

Prior to the DB2 II V8.2, any federated query that joined nicknames with local data was processed serially by the coordinating agent.

In DB2 II V8.2, federated queries that also access local DB2 data can take advantage of intra-partition parallelism by accessing the local data in parallel. Access to the nickname data is still performed serially through the coordinating

agent. Intra-partition parallelism can improve query performance, assuming adequate resources are available.

Note: Intra-partition parallelism is not dependent on the setting of the DB2_FENCED wrapper option. It applies to both trusted and fenced mode operations.

Intra-partition parallelism is enabled as follows:

- ▶ Set the INTRA_PARALLEL database manager configuration parameter to YES.
- ▶ Set the MAX_QUERYDEGREE database manager configuration parameter to a value greater than 1.
- ▶ Set the DFT_DEGREE database configuration parameter to a value greater than 1, or set the special register CURRENT DEGREE. If you set the DFT_DEGREE parameter to ANY, the default level of intra-partition parallelism equals the number of processors on the computer.

Inter-partition parallelism in a DPF environment with local data

Figure A-3 shows inter-partition parallelism support before and after DB2 II V8.2 for federated queries accessing local data in a DPF environment.

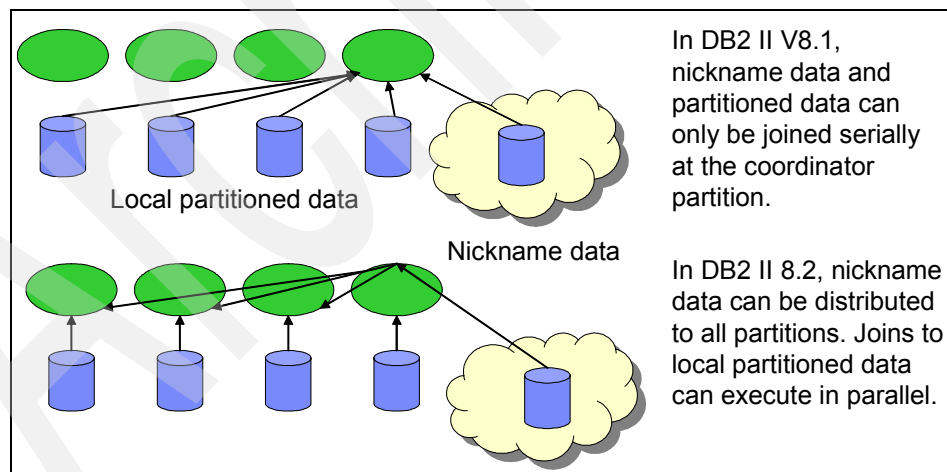


Figure A-3 Inter-partition parallelism in a DPF environment with local data

In a DPF environment with federated queries that reference both local and nickname data, the federated server can distribute the remote data to each of the database partitions.

Prior to DB2 II V8.2, remote nickname data and the local partitioned data were processed serially at a single coordinator partition, as shown in the top portion of Figure A-3 on page 433.

In DB2 V8.2, the federated server may choose to distribute the nickname data to the various database partitions for parallel processing with the local partitioned data, as shown in the bottom portion of Figure A-3 on page 433.

Selection of inter-partition parallelism by the optimizer can result in significant performance gains for federated queries joining nickname data with local partitioned data, since it avoids having all join processing occur serially at the coordinator node.

Such inter-partition parallelism for nickname data is enabled by setting the DB2_FENCED wrapper option to 'Y', as discussed in "Fenced wrappers" on page 430.

Note: The decision to use inter-partition parallelism is made by the optimizer and is cost based, and may not occur if small volumes of local data are involved in the join. In other words, just because the wrapper has the DB2_FENCED option set to 'Y', the optimizer may not estimate that a parallel plan is the lowest cost plan. A serial plan will be selected if it is the lowest cost plan.

Inter-partition parallelism in a DPF environment without local data

Figure A-4 on page 435 shows inter-partition parallelism support before and after DB2 II V8.2 for federated queries that only access nickname data (no local data access involved) in a DPF environment.

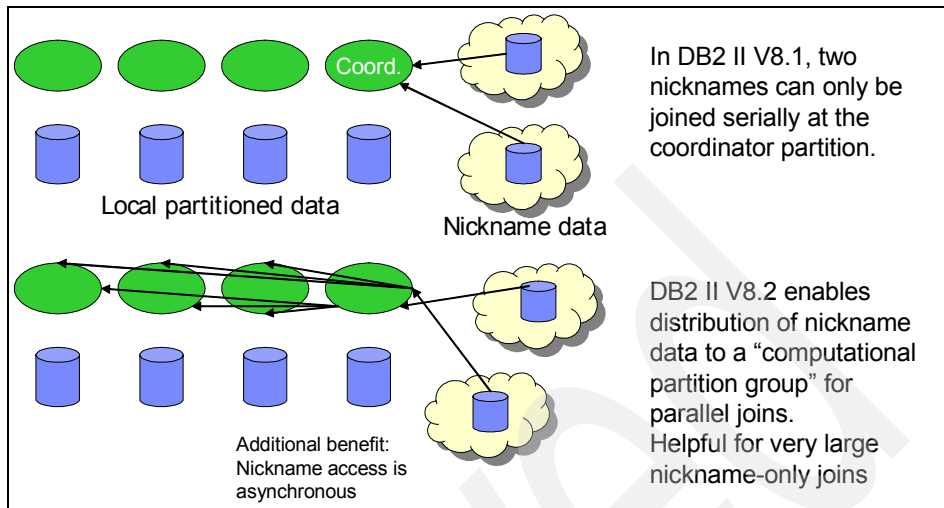


Figure A-4 Inter-partition parallelism in DPF environment with nickname data

In a DPF environment with federated queries that reference only nickname data, if there are operations that are not pushed down to the data sources, the federated server may distribute the remote data to each of the database partitions to take advantage of CPU parallelism and memory of the partitions to process the SQL that was not pushed down. Such a plan will be chosen by the optimizer based on cost and will be selected only if it is the lowest cost plan.

Prior to DB2 II V8.2, remote nickname data was processed serially at the single coordinator partition, as shown in the top portion of Figure A-4.

In DB2 II V8.2, the federated server may choose to distribute nickname data to the various database partitions (defined in a computational partition group¹) for parallel processing, as shown in the bottom portion of Figure A-4.

Selection of inter-partition parallelism can significantly impact performance of federated queries that only join nickname data.

To enable such inter-partition parallelism:

1. A partition group must be defined for the federated database, for example:

```
CREATE DATABASE PARTITION GROUP NDGRP_0_1_2_3 ON DBPARTITIONNUMS(0,1,2,3)
```

¹ A computational partition group defines a set of partitions for the optimizer to use for performing a dynamic redistribution operation for join operations. A computational partition group is a database partition group, other than IBMCATNODEGROUP, that is specified in the system catalog SYSCAT.DBPARTITIONGROUPS.

2. The DB2 registry variable DB2_COMPPARTITIONGROUP must be set to use this partition group:

```
db2set DB2_COMPPARTITIONGROUP = NDGRP_0_1_2_3
```

Updating nickname statistics

The DB2 UDB query optimizer is heavily dependent upon statistics about nickname objects stored in the global catalog in order to generate an optimal access plan. This information is retrieved from the remote objects when the nickname is first created. However, the federated database does not automatically synchronize the global catalog information with that of the remote data sources when changes occur. This can result in the generation of less optimal access plans contributing to poor performance. It is the DBA's responsibility to ensure that the global catalog is kept in sync with information at the remote data sources. Nickname statistics should be updated when the number of records and column distribution values change in tables referenced by nicknames at data sources, and the statistics have been updated at the remote data sources. The federated database DBA should coordinate with DBAs at the data source regarding scheduling or notification when statistics are updated at data sources.

Prior to DB2 II V8.2, the DBA has the choice of disruptive synchronization by dropping and recreating the nickname, or a less disruptive approach of updating the global catalog manually using SQL.

In DB2 II V8.2, the global catalog may be synchronized via the Statistics Update facility in the DB2 Control Center or via a stored procedure SYSPROC.NNSTAT from the command line. You can retrieve the statistics of a single nickname, all nicknames in a DB2 schema on a specific DB2 server definition, all nicknames under a given server, all nicknames under a server with a specific schema, or all the nicknames (when the input parameters are NULL).

The following is an example of using the command line to invoke the SYSPROC.NNSTAT stored procedure for all nicknames on the federated server FEDSERV.

```
CALL SYSPROC.NNSTAT('ORASRV1','NULL','NULL','NULL',?,?,?)
```

Cache tables

In DB2 II Version 8.2, there is a concept similar to MQTs called cache tables, which provide a look-aside capability. A cache table can improve query

performance by accessing a local subset of data instead of accessing data directly from the remote relational data source.

A cache table consists of the following components:

- ▶ A nickname on the federated database system with the same column definitions and same data access as the remote relational data source table.
- ▶ One or more user-maintained MQTs that you define on the nickname. The nickname can contain a subset of high-use data from a remote data source.
- ▶ A user-defined replication schedule that is associated with each materialized query table to keep the local materialized query tables current with your remote data source.

Figure A-5 illustrates the cache table concept.

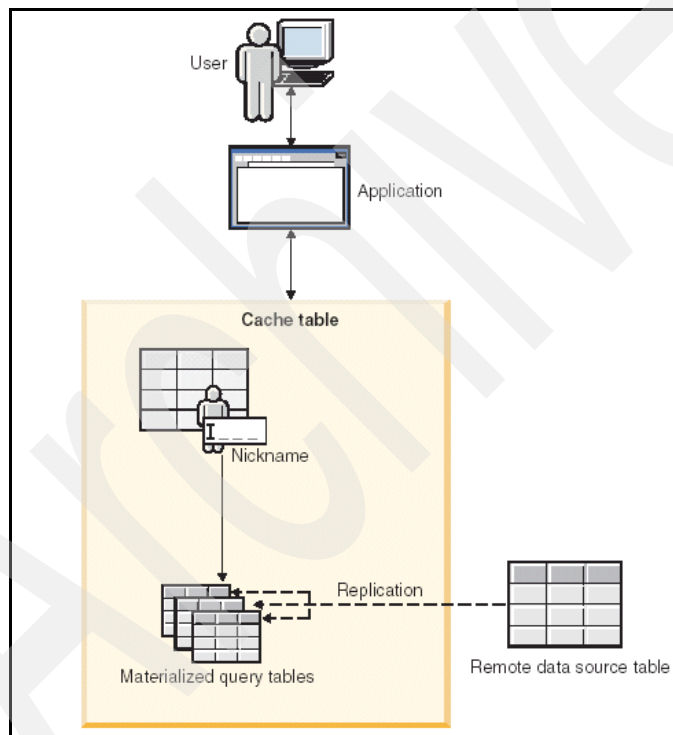


Figure A-5 Cache table concept

The cache table has the same name as the nickname component. A cache table can only be associated with one remote table. The cache table can be a full replica or partial subset of rows from your remote data source. The cache table contains local data that is defined by the MQTs associated with it. There needs to

be a row-to-row relationship between the records in the source table and the records in the MQTs of the cache table.

During query processing, the optimizer directs the query to the cache table or the remote relational data source table.

Cache tables are created using the Cache Table wizard in the DB2 Control Center. Under the icon for a database, select the icon for Cache Tables, and then the Create option. After completing all the entries in the wizard, the wizard will create SQL replication control tables for Capture at the source, the SQL replication control tables for Apply at the federated database, and the staging table at the source. For non-DB2 sources, the wizard also creates the Capture triggers on the source table. Enabling the cache table starts Capture at DB2 sources and Apply at the federated database.

DB2 II uses SQL replication to maintain the data in the cache table MQTs. A staging table is created at the data source for each source table. At DB2 data sources, the SQL Replication Capture process is started and reads the data source log and inserts change records into the staging table. At non-DB2 data sources, Capture triggers are added to the source table to insert records into the staging table whenever the source table is updated. The SQL Replication Apply process at the federated server replicates the changes from the staging table at the data source into the MQTs of the cache table.

Informational constraints

Informational constraints are rules that the optimizer can use to improve performance, that the database manager does not enforce.

With DB2 II V8.2, informational constraints may be defined on nicknames to improve the performance of queries on remote data sources. The following types of informational constraints may be defined for nicknames:

- ▶ Referential constraints
- ▶ Check constraints
- ▶ Functional dependency constraints
- ▶ Primary key constraints
- ▶ Unique constraints

The following is an example of defining information constraints on nickname

```
ALTER NICKNAME account.salary
ADD CONSTRAINT cons1 CHECK( salary > 10000 ) NOT ENFORCED
ENABLE QUERY OPTIMIZATION;
```

Snapshot monitor support

In DB2 II V8.2, the snapshot monitor can capture information about federated data sources and any connected applications at a given time. Snapshots are useful for determining the status of a federated system. Taking snapshots at regular intervals helps in trend analysis for capacity planning, forestalling potential problems, and problem determination.

The dynamic SQL snapshot captures information about query fragments accessing remote data sources, and provides useful information such as the time spent at the remote data source, and the number of rows returned from the data source to the federated server. This information is particularly useful in problem determination and capacity planning.

The monitor switch `STATEMENT` must be set to `ON` to gather this information. The following command captures federated query and query fragment information:

```
get snapshot for dynamic sql on <dbname>
```

Health Center alerts

DB2 Health Center has added two new health indicators to monitor the status of the federated nicknames and remote servers. The federated health indicators are installed when the health monitor is installed. By default, the Health Center does not activate the federated health indicators.

When the health indicators are activated, an alert is issued when the state of a nickname or remote server is not normal. The alert may be viewed using the Health Center or the command line. Federated servers that use AIX, HP-UX, Linux, Microsoft Windows, and Solaris operating systems support the health indicators.

The two health indicators are `db.fed_nicknames_op_status` and `db.fed_servers_op_status`, as follows:

- ▶ `db.fed_nicknames_op_status` is the health indicator for nicknames.
This indicates the aggregate health of all the relational nicknames defined in a database on a DB2 UDB federated server. An alert is generated when a nickname is invalid, and it provides details about the invalid nicknames and recommends actions that can be taken to repair them.
- ▶ `db.fed_servers_op_status` is the health indicator for server definitions.
This indicates the aggregate health of all the federated servers defined in a database on a DB2 UDB federated server. An alert is generated if a remote

server is unavailable, and it provides details about the unavailable servers and recommends actions that can be taken to make them available.

Health checks for nicknames and federated servers are performed at 24-hour intervals. The Health Monitor uses a simple SQL statement (select <col1>, <col2>, etc. from nickname fetch 1 row) to determine:

- ▶ If the data source is available.
- ▶ If the columns of the nickname are still valid. This is how the health check detects if any columns of the source table have been removed, or have different names or data types.

If either of the above problems occurs, an “attention” record is created in the Health Monitor. The health check does not detect if columns have been added to a source table or if indexes have been added or statistics have changed.

DB2 EXPLAIN facility with DB2 Information Integrator

In this appendix, we provide examples of using DB2 EXPLAIN facility's **db2exfmt** tool in a DB2 Information Integrator (DB2 II) Version 8.2 environment, and analyze the access plan of various federated queries involving multiple remote heterogeneous data sources, as well as local data sources.

The topics covered are:

- ▶ Brief review of the DB2 EXPLAIN facility
- ▶ **db2exfmt** overview
- ▶ Federated test environment
- ▶ **db2exfmt** examples involving DB2 II

Brief review of the DB2 EXPLAIN facility

The DB2 EXPLAIN facility allows you to capture information about the environment and the access plan chosen by the optimizer for static or dynamic SQL statements. This information can be used to tune the SQL statements, as well as the database manager configuration to improve performance.

DB2 EXPLAIN captured information includes the following:

- ▶ Sequence of operations to process the query
- ▶ Cost information
- ▶ Predicates and selectivity estimates for each predicate
- ▶ Statistics for all objects referenced in the SQL statement at the time that the EXPLAIN information is captured

The DB2 EXPLAIN facility provides a number of tools to capture, display, and analyze information about the access plans that the DB2 II global optimizer chooses for SQL statements.

While the DB2 EXPLAIN facility tools provide a wealth of valuable information about the access plans and the environment, the following limitations apply:

- ▶ The explain output does not explicitly identify the order of processing of the various access plan operations.
- ▶ The access plan listed in the EXPLAIN output is based on the statistics available at the time of statement compilation. For static SQL this corresponds to bind/prep time, and may not match the actual runtime statistics.
- ▶ EXPLAIN output shows the access path chosen by the optimizer, but does *not* display those paths considered but rejected during cost optimization. For example, if a materialized query table (MQT) was created with the intent of enhancing nickname processing performance, it will not be shown in the EXPLAIN output if the optimizer chooses a different path. It is then left up to the user's knowledge to determine whether the MQT was not chosen because of syntactic limitations (the way the query was written that inhibited the DB2 optimizer for considering the MQT altogether) or because the DB2 optimizer rejected the MQT for cost reasons.

Since alternative access plans that were generated and evaluated by the optimizer are not shown in explain output, you cannot tell from just one explain what operations were restricted from pushdown by pushdown analysis (PDA). Toggling the server option DB2_MAXIMAL_PUSHDOWN between 'N' and 'Y' for consecutive explains should show which operations are restricted from pushdown by PDA, since they will be left in the access graph plan when DB2_MAXIMAL_PUSHDOWN = 'Y'. Operations that

disappear from the access plan graph when DB2_MAXIMAL_PUSHDOWN is toggled from 'N' to 'Y' are operations that PDA allowed to be pushed down but were left to the optimizer to decide whether to push down based on cost. This should be evident when examining the cumulative cost of the plans obtained with DB2_MAXIMAL_PUSHDOWN='N' and the plan obtained with DB2_MAXIMAL_PUSHDOWN='Y'.

- EXPLAIN output does not show settings of DB2 II options such as PUSHDOWN.

Note: Visual Explain may be accessed from two of the GUI admin tools as follows:

- Command editor: On the top menu, select **Access plan**.
- DB2 Control Center: Highlight the icon for a database, and select the **Explain SQL** option.

Table B-1 identifies the various tools that comprise the DB2 EXPLAIN facility, and provides a high-level overview comparison of their capabilities.

Table B-1 DB2 EXPLAIN facility

Desired characteristics	EXPLAIN tables	Visual Explain	db2exfmt	db2expln	dynexpln
GUI interface		Yes			
Text output			Yes	Yes	Yes
“Quick and dirty” Static SQL analysis				Yes	
Static SQL supported	Yes	Yes	Yes	Yes	
Dynamic SQL supported	Yes	Yes	Yes	Yes	Yes (indirectly using db2expln)
CLI applications supported	Yes	Yes	Yes		
Available to DRDA® Application Requesters	Yes				
Detailed optimizer information	Yes	Yes	Yes		
Suited for analysis of multiple statements	Yes		Yes	Yes	Yes

Desired characteristics	EXPLAIN tables	Visual Explain	db2exfmt	db2expln	dynexpln
Information accessible from within an application	Yes				

Access path information is stored in EXPLAIN tables, which can be queried to retrieve the desired information. Either the GUI tool Visual Explain or the text-based **db2exfmt** tool can be used to examine the contents of the EXPLAIN tables.

We will only be using the **db2exfmt** tool in our examples. Therefore, we will only briefly describe the DB2 EXPLAIN tables and the results of **db2exfmt** in the following sections.

Please refer to *IBM DB2 UDB Administration Guide: Performance*, SC09-4821-00, for complete details about all of the tools mentioned in Table B-1 on page 443.

Table B-2 lists the main EXPLAIN tables, while Figure B-1 on page 446 shows the relationships between them.

Table B-2 EXPLAIN tables

Table name	Description
EXPLAIN_INSTANCE	The main control table for all EXPLAIN information. Each row in the EXPLAIN tables is explicitly linked to one unique row in this table.
EXPLAIN_STATEMENTS	This table stores the EXPLAIN snapshot if it was requested. Data is stored as Binary Large Object (BLOB), which contains the internal representation of the access plan and decision criteria used by the DB2 optimizer. A row in the EXPLAIN_INSTANCE refers to multiple rows in this table.
EXPLAIN_OPERATORS	This table contains all the operators needed to satisfy the query. The types of operators include FETCH, GRPBY, IXSCAN, MSJOIN, NLJOIN, RIDSCN, SORT, TBSCAN, TEMP or UNIQUE.
EXPLAIN_ARGUMENTS	This table contains the information for each operator; for example, for a SORT operator, arguments such as number of rows expected to be sorted are collected.
EXPLAIN_OBJECTS	This table identifies the data objects required by the access plan. Types of objects are indexes, tables, views, nicknames, and table functions.

Table name	Description
EXPLAIN_STREAM	This table represents the input and output data stream between operators and objects, for example, the number of columns represented and an estimate of the cardinality.
EXPLAIN_PREDICATE	This table identifies which predicates are applied to a specific operator.
ADVISE_WORKLOAD	This table allows users to describe a workload to the database. Each row in the table represents an SQL statement in the workload and is described by an associated frequency. The db2advis tool uses this table to collect and store work and information.
ADVISE_INDEX	<p>This table stores information about recommended indexes. The table can be populated by the SQL compiler, the db2advis utility, or a user. This table is used in two ways:</p> <ul style="list-style-type: none"> ▶ To get recommended indexes ▶ To evaluate indexes based on input about proposed indexes

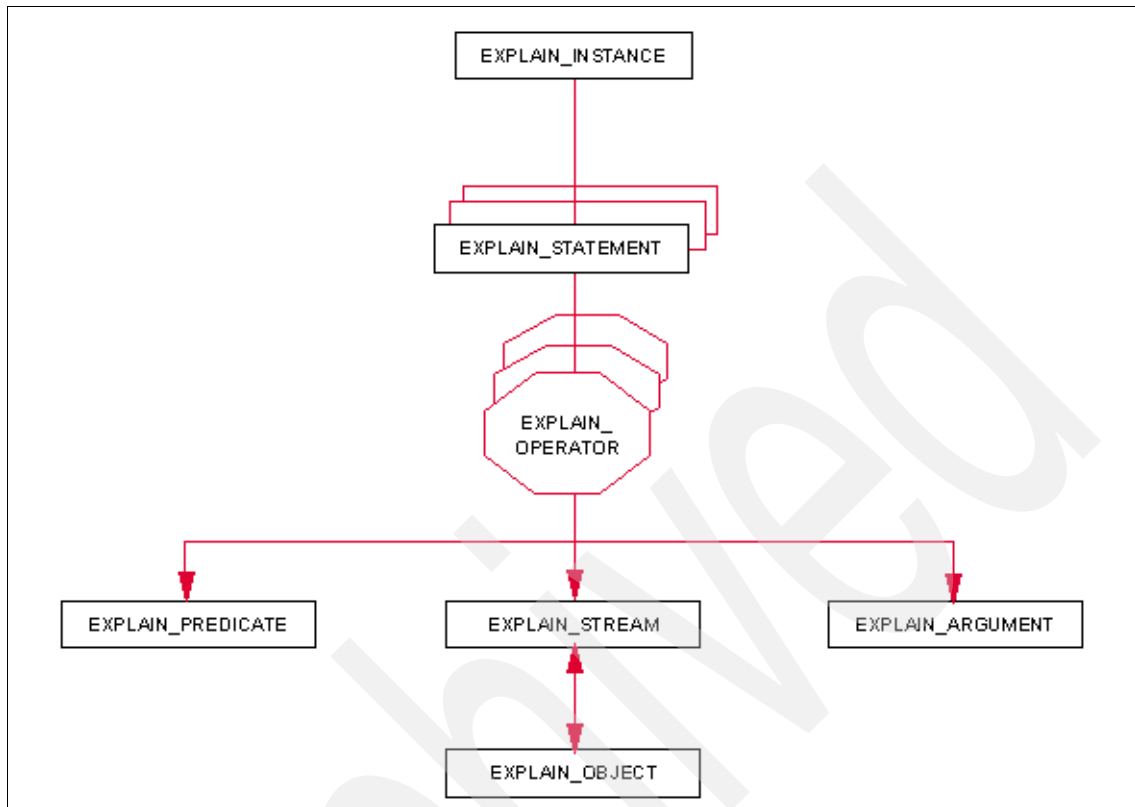


Figure B-1 Relationship of the main EXPLAIN tables

Note: EXPLAIN tables can be created by the issuing the **db2 -tf EXPLAIN.DDL** command, or automatically by the DB2 Control Center. The EXPLAIN.DDL file is located in the **\$HOME/sqllib/misc** directory on UNIX/Linux, where \$HOME is the home directory of the DB2/II instance owner and is located in **c:\Program Files\IBM\SQLLIB\misc** on Windows.

Table B-3 lists the operators that may appear in EXPLAIN output.

Table B-3 Operators in the access plan

Operator	Function
BTQ	Broadcast Table Queue broadcasts data to several partitions.
DELETE	Deletes rows from a table.

Operator	Function
DTQ	Directed Table Queue transfers data to a specific partition.
EISCAN	Scans a user-defined index to produce a reduced stream of rows.
FETCH	Fetches columns from a table using specific record identifier.
FILTER	Represents the application of residual predicates.
GRPBY	Groups rows by common values of designated columns or functions.
HSJOIN	Represents a hash join, where two or more tables are hashed on join columns.
INSERT	Inserts rows into a table.
IXAND	ANDs together the row identifiers (RIDs) from two or more index scans.
IXSCAN	Scans an index of a table with optional start/stop conditions, producing an ordered stream of rows.
LTQ	Local Table Queue. Transfers data between local agents.
LMTQ	Local Merge Table Queue. Merges data transferred between local agents.
MBTQ	Merging Broadcast Table Queue.
MDTQ	Merging Directed Table queue.
MSJOIN	Represents a merge join, where both outer and inner tables must be in join-predicate order.
NLJOIN	Represents a nested loop join that accesses an inner table once for each row of the outer table.
RETURN	Represents the return of data from the query to the user.
RIDSCAN	Scans a list of row identifiers (RIDs) obtained from one or more indexes.

Operator	Function
RPD	For nonrelational wrappers, it shows the simulated SQL operation that the nonrelational wrapper will be asked to perform.
SHIP	Retrieves data from a remote database source. Used in federated systems.
SORT	Sorts rows in the order of specified columns, and optionally eliminates duplicate entries.
TBSCAN	Retrieves rows by reading all required data directly from the data pages.
TEMP	Stores data in a temporary table to be read back out (possibly multiple times).
TQUEUE	Transfers table data between agents.
UNION	Concatenates streams of rows from multiple tables.
UNIQUE	Eliminates rows with duplicate values, for specified columns.
UPDATE	Updates rows in a table.

db2exfmt overview

This tool is used to format the contents of the EXPLAIN tables. Using the sample federated SQL statement shown in Example B-1, which reports the amount of business that was billed, shipped, and returned during a period of interest, we will briefly describe the output of this tool.

Example: B-1 Sample federated SQL statement

```

SELECT
    L_RETURNFLAG,
    L_LINESTATUS,
    SUM(L_QUANTITY) AS SUM_QTY,
    SUM(L_EXTENDEDPRI) AS SUM_BASE_PRICE,
    SUM(L_EXTENDEDPRI * (1-L_DISCOUNT)) AS SUM_DISC_PRICE,
    SUM(L_EXTENDEDPRI * (1-L_DISCOUNT) * (1+L_TAX)) AS SUM_CHARGE,
    AVG(L_QUANTITY) AS AVG_QTY,
    AVG(L_EXTENDEDPRI) AS AVG_PRICE,
    AVG(L_DISCOUNT) AS AVG_DISC,
    COUNT(*) AS COUNT_ORDER

```

```

FROM
    ORA.LINEITEM

WHERE
    L_SHIPDATE <= DATE ('1998-12-01') - 90 DAYS
GROUP BY
    L_RETURNFLAG,
    L_LINESTATUS
ORDER BY
    L_RETURNFLAG,
    L_LINESTATUS
;

```

This SQL statement needs to be explained in order to populate the EXPLAIN tables.

Once the EXPLAIN tables have been populated, the following command is used to generate **db2exfmt** output for this SQL statement:

```
db2exfmt -d feddb -l -s ora -o db2exfmt.out
```

The output of **db2exfmt** is divided into five sections, as follows:

- ▶ Explain Instance section
- ▶ SQL Statement section
- ▶ Access plan graph
- ▶ Operator Details section
- ▶ Object section

Each of these sections is described below using fragments of **db2exfmt** output corresponding to the particular section.

EXPLAIN INSTANCE section

The fragment corresponding to this section is shown in Example B-2.

Example: B-2 EXPLAIN INSTANCE section

```
***** EXPLAIN INSTANCE *****
```

```

DB2_VERSION: 08.02.0
SOURCE_NAME: TOOL1E00
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-06-15-03.11.32.302527
EXPLAIN_REQUESTER: KAWA

```

Database Context:

```

-----
Parallelism: None
CPU Speed: 4.723442e-07
Comm Speed: 100
Buffer Pool size: 75000
Sort Heap size: 20000
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 1020

Package Context:
-----
SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

```

The EXPLAIN INSTANCE section lists overall DB2 instance and database parameter information.

Besides identification details such as the DB2 code base level, requestor name, and time of request, this section also provides environmental settings in which the SQL compiler optimized the query. Information provided includes total buffer pool size (sum of all buffer pools including temporary table spaces), sort heap size, type of query parallelism, dynamic or static SQL, optimization level, and isolation levels. The CPU and Communications speeds are used by the optimizer to compare the capabilities of the remote data source servers during cost optimization to determine predicate pushdown. This environmental settings are critical to understanding the optimizer's decision in arriving at a particular access plan.

SQL STATEMENT section

Example B-3 lists the section that provides additional information about the SQL statement, and shows how DB2 Query Rewrite has rewritten the SQL *before* it is processed by pushdown analysis. Using global semantics, query rewrite transforms SQL statements into forms that can be optimized more easily, and as a result can improve the possible access paths. Queries might be rewritten in multiple ways, including operation merging, operation movement, and predicate translation.

Example: B-3 STATEMENT section

```

----- STATEMENT 1 SECTION 1 -----
QUERYNO: 1

```


QUERYTAG:
 Statement Type: Select
 Updatable: No
 Deletable: No
 Query Degree: 1

Original Statement:

```

-----
SELECT L_RETURNFLAG, L_LINESTATUS, SUM(L_QUANTITY) AS SUM_QTY,
       SUM(L_EXTENDEDPRICE) AS SUM_BASE_PRICE, SUM(L_EXTENDEDPRICE *
       (1-L_DISCOUNT)) AS SUM_DISC_PRICE, SUM(L_EXTENDEDPRICE *
       (1-L_DISCOUNT) * (1+L_TAX)) AS SUM_CHARGE, AVG(L_QUANTITY) AS AVG_QTY,
       AVG(L_EXTENDEDPRICE) AS AVG_PRICE, AVG(L_DISCOUNT) AS AVG_DISC,
       COUNT(*) AS COUNT_ORDER
FROM ora.LINEITEM
WHERE L_SHIPDATE <= DATE ('1998-12-01') - 90 DAYS
GROUP BY L_RETURNFLAG, L_LINESTATUS
ORDER BY L_RETURNFLAG, L_LINESTATUS
  
```

Optimized Statement:

```

-----
SELECT Q3.$C0 AS "L_RETURNFLAG", Q3.$C1 AS "L_LINESTATUS", Q3.$C2 AS
       "SUM_QTY", Q3.$C3 AS "SUM_BASE_PRICE", Q3.$C4 AS "SUM_DISC_PRICE",
       Q3.$C5 AS "SUM_CHARGE", (Q3.$C2 / Q3.$C6) AS "AVG_QTY", (Q3.$C3 /
       Q3.$C6) AS "AVG_PRICE", (Q3.$C7 / Q3.$C6) AS "AVG_DISC", Q3.$C6 AS
       "COUNT_ORDER"
FROM
  (SELECT Q2.$C0, Q2.$C1, SUM(Q2.$C2), SUM(Q2.$C3), SUM((Q2.$C3 * (1 -
       Q2.$C4))), SUM(((Q2.$C3 * (1 - Q2.$C4)) * (1 + Q2.$C5))), COUNT(*)
       ), SUM(Q2.$C4)
  FROM
    (SELECT Q1.L_RETURNFLAG, Q1.L_LINESTATUS, Q1.L_QUANTITY,
           Q1.L_EXTENDEDPRICE, Q1.L_DISCOUNT, Q1.L_TAX
     FROM ORA.LINEITEM AS Q1
     WHERE (Q1.L_SHIPDATE <= '09/02/1998')) AS Q2
  GROUP BY Q2.$C1, Q2.$C0) AS Q3
ORDER BY Q3.$C0, Q3.$C1
  
```

Two versions of the text of the SQL statement are recorded for each statement explained.

- ▶ The original statement is the code that the SQL compiler receives from the application. The original SQL statement in **db2exfmt** shows the SQL submitted by the user. If the user SQL included any views, their names appear in the original SQL statement.
- ▶ The Optimized statement is reverse translated from the internal compiler representation of the query.

In case of views, the Optimized statement shows that query rewrite has obtained the text of any views in the user's SQL statement and combined it with the SQL submitted by the user so the Optimized statement shows all the real tables and nicknames (no views) that will need to be accessed to execute the user's SQL statement.

Note: Although the translation looks similar to other SQL statements, it does *not* necessarily follow correct SQL syntax, nor does it necessarily reflect the actual content of the internal representation as a whole. This translation is provided only to allow you to understand the SQL context in which the SQL optimizer chose the access plan.

To understand how the SQL compiler has rewritten your query for better optimization, compare the user-written statement text to the internal representation of the SQL statement. The rewritten statement also shows you other elements in the environment affecting the SQL statement such as triggers and constraints. For example, $\$Cn$ is the name of a derived column, where n represents an integer value.

Please refer to the *IBM DB2 UDB Administration Guide: Performance*, SC09-4821-00, for a complete description of various elements in the environment affecting the SQL statement.

Attention: The optimized statement does not necessarily reflect the actual statement that is executed. For instance, if Query Rewrite caused a routing to occur to a non-aggregate materialized query table (MQT), then this would not be reflected in the optimized statement. However, a routing to an aggregate MQT would be reflected in the optimized statement. The Access Plan section describes the actual execution plan chosen by the optimizer.

An MQT may be used because of decisions by either Query Rewrite or the optimizer. Query Rewrite has internal rules by which it determines whether an MQT should be substituted for a nickname, such as on the basis of informational constraints.

If Query Rewrite makes the substitution:

- ▶ The original SQL statement contains the nickname.
- ▶ The optimized SQL contains the MQT name.
- ▶ The access plan contains the MQT name.

If Query Rewrite internal rules do not cause it to substitute the MQT for the nickname, but the optimizer determines that the MQT can be substituted for the

nickname and the plan using the MQT is lower cost than the plan using the nickname, then:

- ▶ The original SQL statement contains the nickname.
- ▶ The optimized SQL contains the nickname.
- ▶ The access plan contains the MQT name.

If Query Rewrite internal rules do not cause it to substitute the MQT for the nickname, and the optimizer determines that the MQT can be substituted for the nickname but the plan using the MQT is higher cost than the plan using the nickname, then:

- ▶ The original SQL statement contains the nickname.
- ▶ The optimized SQL contains the nickname.
- ▶ The access plan contains the nickname (the MQT is not used).

If Query Rewrite internal rules do not cause it to substitute the MQT for the nickname, and the optimizer determines that the MQT cannot be substituted for the

- ▶ The original SQL statement contains the nickname.
- ▶ The optimized SQL contains the nickname.
- ▶ The access plan contains the nickname (the MQT is not used).

Access plan graph

Example B-4 shows this section of the output as a graphical representation of the plan created to execute the SQL statement.

Example: B-4 Access Plan section

Access Plan:

Total Cost: 8.22466e+06

Query Degree:1

```

      Rows
      RETURN
      ( 1)
      Cost
      I/O
      |
      6
      SHIP
      ( 2)
      8.22466e+06
      2.08104e+06
      |
  
```

5.99861e+07
NICKNM: ORA
LINEITEM

At the top of the access plan graph is the total query cost in timerons¹ and the query degree of parallelism used.

The following points can be noted about the graph:

- ▶ The graph is made up of objects (nickname ORA.LINEITEM)—operators (RETURN and SHIP) that are connected by lines. Table B-3 on page 446 defines the function of the operators shown in the **db2exfmt** output.
- ▶ Each operator has a reference number included in parentheses, for example, the reference number for the SHIP operator is 2. This reference number does *not* imply the operator execution order.
- ▶ Values above and below the object and operators represent cardinality and costs, as described below.

The access plan graph should be read as follows:

- ▶ Read the graph from the bottom up since it represents the order of the sub-operations as they would be executed. The RETURN operator at the top represents the query result.
- ▶ For each operator, the legend for the numbers above and below are explained in the operator with the title 'RETURN'.
 - The number above the operator is the number of rows estimated to be involved in this sub-operation.
 - The reference number for finding details of this operator is in the section below the access plan graph (in parenthesis).
 - The number immediately below the operator is the cumulative estimated cost expressed in timerons for the sub-operation and the sub-operations that precede it reading from the bottom up. The incremental estimated cost of an operation is the number below the operator minus the number below the operator that precedes it.
 - The second number below an operator is an estimated cumulative I/O cost of the operator.
- ▶ For an object such as the ORA.LINEITEM nickname, the value above its name represents the cardinality of the object as derived from the CARD column of the SYSCAT.TABLES view in the DB2 II catalog.

¹ A timeron is an abstract unit of measure that does not directly equate to any actual elapsed time, but gives a relative estimate of the resources required by the database manager to execute an access plan. The resources calculated in the estimate include weighted CPU, I/O, and remote communication costs.

The SHIP operator is unique to a DB2 II environment and indicates that an SQL statement is sent to a remote federated data source for processing. All operators and objects above a SHIP operator indicate operations that are performed locally at the federated server and not pushed down to the remote federated data source. The operations performed at the federated server appear as operators in the **db2exfmt** access plan graph, and operations pushed down to data sources do not appear in the access plan graph. The operations pushed down to the data sources can be found in the RMTQTX field in the detail sections for the SHIP operators below the access plan graph in the **db2exfmt** output.

The access plan shown in Example B-4 on page 453 estimates that a total of six rows will be returned to the requestor at an estimated cost of 8.22466e+06 timerons. It lists the cardinality of the ORA.LINEITEM nickname to be 5.99861e+07 rows, and that the SHIP operator returns six rows to the federated server at the estimated cost of 8.22466e+06 timerons. Since there are no objects or operators (other than RETURN) above the SHIP operator, it indicates that all predicates, GROUP BY and ORDER BY were pushed down to the remote federated data source.

OPERATOR DETAILS section

Example B-5 shows this section containing detailed information for each operator listed in the access plan graph.

Example: B-5 OPERATOR DETAILS section

```

2) SHIP : (Ship)
  Cumulative Total Cost: 8.22466e+06
  Cumulative CPU Cost: 1.74852e+11
  Cumulative I/O Cost: 2.08104e+06
  Cumulative Re-Total Cost: 8.1887e+06
  Cumulative Re-CPU Cost: 9.87231e+10
  Cumulative Re-I/O Cost: 0
  Cumulative First Row Cost: 8.22466e+06
  Estimated Bufferpool Buffers: 0
  Remote communication cost:11.8594

Arguments:
-----
CSERQY : (Remote common subexpression)
  FALSE
DSTSEVER: (Destination (ship to) server)
  - (NULL).
RMTQTX : (Remote statement)
  SELECT AO."L_RETURNFLAG", AO."L_LINESTATUS", SUM( AO."L_QUANTITY"),
SUM( AO."L_EXTENDEDPRI" ), SUM( (AO."L_EXTENDEDPRI" * (1 -
(AO."L_DISCOUNT")))), SUM( ((AO."L_EXTENDEDPRI" * (1 - (AO."L_DISCOUNT")))) *

```

```
(1 + AO."L_TAX"))), (SUM( AO."L_QUANTITY" ) / COUNT(*)), (SUM(
AO."L_EXTENDEDPRICE" ) / COUNT(*)), (SUM( AO."L_DISCOUNT" ) / COUNT(*)), COUNT(*)
FROM "IITEST"."LINEITEM" AO WHERE (AO."L_SHIPDATE" <= TO_DATE('19980902
000000','YYYYMMDD HH24MISS')) GROUP BY AO."L_RETURNFLAG", AO."L_LINESTATUS"
ORDER BY 1 ASC, 2 ASC
```

```
SRCSEVER: (Source (ship from) server)
ORASERV
STREAM : (Remote stream)
TRUE
```

Input Streams:

1) From Object ORA.LINEITEM

```
Estimated number of rows: 5.99861e+07
Number of columns: 8
Subquery predicate ID: Not Applicable
```

Column Names:

```
-----
+Q1.$RID$+Q1.L_TAX+Q1.L_DISCOUNT
+Q1.L_EXTENDEDPRICE+Q1.L_QUANTITY
+Q1.L_LINESTATUS+Q1.L_RETURNFLAG+Q1.L_SHIPDATE
```

Output Streams:

2) To Operator #1

```
Estimated number of rows: 6
Number of columns: 12
Subquery predicate ID: Not Applicable
```

Column Names:

```
-----
+Q3.L_RETURNFLAG(A)+Q3.L_LINESTATUS(A)
+Q4.COUNT_ORDER+Q4.AVG_DISC+Q4.AVG_PRICE
+Q4.AVG_QTY+Q4.SUM_CHARGE+Q4.SUM_DISC_PRICE
+Q4.SUM_BASE_PRICE+Q4.SUM_QTY+Q4.L_LINESTATUS
+Q4.L_RETURNFLAG
+NONE
```

The reference number in the access plan graph is used to identify the details of a specific operator in this section.

Note: We have only shown the SHIP operator portion of this section in Example B-5 and not the RETURN operator.

Besides the cumulative cost information provided for each operator, additional categories of information are also listed for specific operators as follows:

- Arguments list the options for the particular operator.

For a SHIP operator:

- SRCSERVER defines the remote data source server name as defined in the SERVERNAME column of the SYSCAT.SERVERS view in the DB2 II catalog.
- RMTQTXTEXT shows the SQL statement to be sent to the data source, in the remote statement text.

Compare the RMTQTXTEXT with the original SQL statement and the optimized SQL statement to determine operations/functions not pushed down to the remote federated data source.

- Input streams describe the source of the rows processed by the operator, including the estimated number of rows, and number and names of the columns involved.
- Output streams describe the destination of the rows processed by the operator, including the estimated number of rows, and number and names of the columns involved.

Objects section

Example B-6 shows the objects used by the access plan and includes tables, indexes, and nicknames. For each of these objects, it lists statistics information as well as other estimates used by the optimizer in arriving at the access plan.

Example: B-6 Objects section

Objects Used in Access Plan:

Schema: MSS
Name: NATION
Type: Nickname
Time of creation: 2004-06-18-13.29.41.931813
Last statistics update:
Number of columns: 4
Number of rows: 25
Width of rows: 48
Number of buffer pool pages: 2
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node

Complete db2exfmt output

Example B-7 is a complete listing of the **db2exfmt** output for the sample federated statement.

Example: B-7 db2exfmt output for the sample federated statement

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

***** EXPLAIN INSTANCE *****

DB2_VERSION: 08.02.0
SOURCE_NAME: TOOL1E00
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-06-15-03.11.32.302527
EXPLAIN_REQUESTER: KAWA

Database Context:

Parallelism: None
CPU Speed: 4.723442e-07
Comm Speed: 100
Buffer Pool size: 75000
Sort Heap size: 20000
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 1020

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 1 -----


```

QUERYNO: 1
QUERYTAG:
Statement Type: Select
Udatable: No
Deletable: No
Query Degree: 1

```

Original Statement:

```

-----
SELECT L_RETURNFLAG, L_LINESTATUS, SUM(L_QUANTITY) AS SUM_QTY,
       SUM(L_EXTENDEDPRICE) AS SUM_BASE_PRICE, SUM(L_EXTENDEDPRICE *
       (1-L_DISCOUNT)) AS SUM_DISC_PRICE, SUM(L_EXTENDEDPRICE *
       (1-L_DISCOUNT) * (1+L_TAX)) AS SUM_CHARGE, AVG(L_QUANTITY) AS AVG_QTY,
       AVG(L_EXTENDEDPRICE) AS AVG_PRICE, AVG(L_DISCOUNT) AS AVG_DISC,
       COUNT(*) AS COUNT_ORDER
FROM ora.LINEITEM
WHERE L_SHIPDATE <= DATE ('1998-12-01') - 90 DAYS
GROUP BY L_RETURNFLAG, L_LINESTATUS
ORDER BY L_RETURNFLAG, L_LINESTATUS

```

Optimized Statement:

```

-----
SELECT Q3.$C0 AS "L_RETURNFLAG", Q3.$C1 AS "L_LINESTATUS", Q3.$C2 AS
       "SUM_QTY", Q3.$C3 AS "SUM_BASE_PRICE", Q3.$C4 AS "SUM_DISC_PRICE",
       Q3.$C5 AS "SUM_CHARGE", (Q3.$C2 / Q3.$C6) AS "AVG_QTY", (Q3.$C3 /
       Q3.$C6) AS "AVG_PRICE", (Q3.$C7 / Q3.$C6) AS "AVG_DISC", Q3.$C6 AS
       "COUNT_ORDER"
FROM
  (SELECT Q2.$C0, Q2.$C1, SUM(Q2.$C2), SUM(Q2.$C3), SUM((Q2.$C3 * (1 -
    Q2.$C4))), SUM(((Q2.$C3 * (1 - Q2.$C4)) * (1 + Q2.$C5))), COUNT(*
    ), SUM(Q2.$C4)
  FROM
    (SELECT Q1.L_RETURNFLAG, Q1.L_LINESTATUS, Q1.L_QUANTITY,
      Q1.L_EXTENDEDPRICE, Q1.L_DISCOUNT, Q1.L_TAX
    FROM ORA.LINEITEM AS Q1
    WHERE (Q1.L_SHIPDATE <= '09/02/1998')) AS Q2
  GROUP BY Q2.$C1, Q2.$C0) AS Q3
ORDER BY Q3.$C0, Q3.$C1

```

Access Plan:

```

-----
Total Cost: 8.22466e+06
Query Degree:1

```

```

Rows
RETURN
( 1)
Cost

```

```

      I/O
      |
      6
    SHIP
    ( 2)
8.22466e+06
2.08104e+06
      |
5.99861e+07
NICKNM: ORA
LINEITEM

```

1) RETURN: (Return Result)

```

Cumulative Total Cost: 8.22466e+06
Cumulative CPU Cost: 1.74852e+11
Cumulative I/O Cost: 2.08104e+06
Cumulative Re-Total Cost: 8.1887e+06
Cumulative Re-CPU Cost: 9.87231e+10
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 8.22466e+06
Estimated Bufferpool Buffers: 0
Remote communication cost:11.8594

```

Arguments:

```

-----
BLDLEVEL: (Build level)
          DB2 v8.1.1.64 : s040509
ENVVAR   : (Environment Variable)
          DB2_EXTENDED_OPTIMIZATION = ON
STMTHEAP: (Statement heap size)
          8192

```

Input Streams:

2) From Operator #2

```

Estimated number of rows: 6
Number of columns: 12
Subquery predicate ID: Not Applicable

```

Column Names:

```

-----
+Q3.L_RETURNFLAG(A)+Q3.L_LINESTATUS(A)
+Q4.COUNT_ORDER+Q4.AVG_DISC+Q4.AVG_PRICE
+Q4.AVG_QTY+Q4.SUM_CHARGE+Q4.SUM_DISC_PRICE
+Q4.SUM_BASE_PRICE+Q4.SUM_QTY+Q4.L_LINESTATUS

```

+Q4.L_RETURNFLAG

2) SHIP : (Ship)

Cumulative Total Cost: 8.22466e+06
Cumulative CPU Cost: 1.74852e+11
Cumulative I/O Cost: 2.08104e+06
Cumulative Re-Total Cost: 8.1887e+06
Cumulative Re-CPU Cost: 9.87231e+10
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 8.22466e+06
Estimated Bufferpool Buffers: 0
Remote communication cost:11.8594

Arguments:

CSERQY : (Remote common subexpression)

FALSE

DSTSEVER: (Destination (ship to) server)

- (NULL).

RMTQTX : (Remote statement)

SELECT AO."L_RETURNFLAG", AO."L_LINESTATUS", SUM(AO."L_QUANTITY"),
SUM(AO."L_EXTENDEDPRICE"), SUM((AO."L_EXTENDEDPRICE" * (1 -
(AO."L_DISCOUNT")))), SUM(((AO."L_EXTENDEDPRICE" * (1 - (AO."L_DISCOUNT")) *
(1 + AO."L_TAX"))), (SUM(AO."L_QUANTITY") / COUNT(*)), (SUM(
AO."L_EXTENDEDPRICE") / COUNT(*)), (SUM(AO."L_DISCOUNT") / COUNT(*)), COUNT(*)
FROM "IITEST"."LINEITEM" AO WHERE (AO."L_SHIPDATE" <= TO_DATE('19980902
000000','YYYYMMDD HH24MISS')) GROUP BY AO."L_RETURNFLAG", AO."L_LINESTATUS"
ORDER BY 1 ASC, 2 ASC

SRCSEVER: (Source (ship from) server)

ORASERV

STREAM : (Remote stream)

TRUE

Input Streams:

1) From Object ORA.LINEITEM

Estimated number of rows: 5.99861e+07

Number of columns: 8

Subquery predicate ID: Not Applicable

Column Names:

+Q1.\$RID\$+Q1.L_TAX+Q1.L_DISCOUNT

+Q1.L_EXTENDEDPRICE+Q1.L_QUANTITY

+Q1.L_LINESTATUS+Q1.L_RETURNFLAG+Q1.L_SHIPDATE

Output Streams:

2) To Operator #1

Estimated number of rows: 6

Number of columns: 12

Subquery predicate ID: Not Applicable

Column Names:

+Q3.L_RETURNFLAG(A)+Q3.L_LINESTATUS(A)
+Q4.COUNT_ORDER+Q4.AVG_DISC+Q4.AVG_PRICE
+Q4.AVG_QTY+Q4.SUM_CHARGE+Q4.SUM_DISC_PRICE
+Q4.SUM_BASE_PRICE+Q4.SUM_QTY+Q4.L_LINESTATUS
+Q4.L_RETURNFLAG

Objects Used in Access Plan:

Schema: ORA

Name: LINEITEM

Type: Nickname

Time of creation: 2004-06-11-21.33.10.537308

Last statistics update: 2004-06-11-22.32.45.978864

Number of columns: 16

Number of rows: 59986052

Width of rows: 64

Number of buffer pool pages: 2081039

Distinct row values: No

Tablespace name:

Tablespace overhead: 24.100000

Tablespace transfer rate: 0.900000

Source for statistics: Single Node

Prefetch page count: 32

Container extent page count: 32

Federated test environment

We used the same federated test environment as described in 4.4.1, “Federated test environment” on page 167, for generating and analyzing **db2exfmt** output for a number of federated queries.

Note: We had multiple instances on Jamesbay that we used for our **db2exfmt** examples. A 32-bit instance with local non-DPF data was used for SQL server access, while a 64-bit instance with local DPF data was used in the fenced and trusted wrapper examples.

db2exfmt examples involving DB2 II

In the following sections, we explain a number of federated queries using various combinations of DB2 database manager configuration, DB2 II server and nickname options, and format the explain output using **db2exfmt**. A brief review and analysis of the access plan in the **db2exfmt** output is provided for the following federated queries:

- ▶ Join of nicknames referencing Oracle and SQL Server
- ▶ INTRA_PARALLEL = YES (intra-partition enabled)
- ▶ Database Partition Feature (DPF) with FENCED = N
- ▶ Database Partition Feature (DPF) with FENCED = Y
- ▶ DB2_MAXIMAL_PUSHDOWN = N
- ▶ DB2_MAXIMAL_PUSHDOWN = Y
- ▶ SQL INSERT/UPDATE/DELETE

Depending on the complexity of the SQL statement, the **db2exfmt** output can be quite voluminous. For a performance perspective, the main focus areas and corresponding EXPLAIN indicator/operator of interest in the output are listed in Table B-4. The sequence of focus areas listed in Table B-4 does not necessarily imply any particular priority for problem analysis. Table B-5 on page 464 lists the main DB2 II server options.

Table B-4 db2exfmt output focus areas

Focus area	Explain indicator/operator	Tuning controls
DB2 II catalog currency -- nickname statistics and index specifications	Default "1000" rows	SYSPROC.NNSTAT CREATE INDEX
Pushdown	SHIP, RMTQTXT, FILTER, GRPBY	Server options PUSHDOWN, DB2_MAXIMAL_PUS HDOWN, CPU_RATIO, IO_RATIO, COMM_RATE,
Join method	HSJOIN, MSJOIN, NLJOIN	Buffer pool, sort heap

Focus area	Explain indicator/operator	Tuning controls
Intra-partition parallelism	LTQ, LMTQ	DFT_DEGREE
Inter-partition parallelism	DTQ, BTQ, MDTQ	CPG, Wrapper Type

Table B-5 DB2 II server options

Option	Description	Default
COMM_RATE	Specifies the communication rate between the federated server and the data source server. Expressed in megabytes per second.	2
CPU_RATIO	Indicates how much faster or slower a data source CPU runs than the federated server.	'1.0'
DB2_MAXIMAL_PUSHDOWN	Specifies the primary criteria that the query optimizer uses when choosing an access plan.	'N'
IO_RATIO	Denotes how much faster or slower a data source I/O system runs than the federated server I/O system.	'1.0'
LOGIN_TIMEOUT	Specifies the number of seconds for the DB2 federated server to wait for a response from Sybase Open Client to the login request.	'0'
PACKET_SIZE	Specifies the packet size of the Sybase interfaces file in bytes.	
PLAN_HINTS	Specifies whether plan hints are to be enabled. Plan hints are statement fragments that provide extra information for data source optimizers.	'N'

Option	Description	Default
PUSHDOWN	'Y' - DB2 UDB will consider letting the data source evaluate operations. 'N' - DB2 UDB will send the data source SQL statements that include only SELECT with column names. Predicates (such as WHERE=) column and scalar functions (such as MAX and MIN), sorts (such as ORDER BY or GROUP BY), and joins will not be included in any SQL sent to the data source.	'Y'
TIMEOUT	Specifies the number of seconds the DB2 federated server will wait for a response from Sybase Open Client for any SQL statement. The value of <i>seconds</i> is a positive whole number in DB2 UDB's integer range. The timeout value that you specify depends on which wrapper you are using. The default behavior of the TIMEOUT option for the Sybase wrappers is 0, which causes DB2 UDB to wait indefinitely for a response. BioRS: Specifies the time, in minutes, that the BioRS wrapper should wait for a response from the BioRS server. The default value is 10.	'0'

Option	Description	Default
VARCHAR_NO_TRAILING_BLANKS	<p>This option applies to data sources that have variable character data types that do not pad the length with trailing blanks during comparison.</p> <p>Some data sources, such as Oracle, do not have blank-padded character comparison semantics that return the same results as the DB2 for Linux, UNIX, and Windows comparison semantics. Set this option when you want it to apply to all the VARCHAR and VARCHAR2 columns in the data source objects that will be accessed from the designated server. This includes views.</p> <p>Y - Trailing blanks are absent from these VARCHAR columns, or the data source has blank-padded character comparison semantics that are similar to the semantics on the federated server. The federated server pushes down character comparison operations to the data source for processing.</p> <p>N Trailing blanks are present in these VARCHAR columns and the data source has blank-padded character comparison semantics that are different than the federated server.</p>	N for affected data sources.

Each of the focus areas identified in Table B-4 on page 463 are covered briefly:

► DB2 II catalog currency

The DB2 optimizer relies on accurate statistics about nickname objects (table and related indexes) in order to generate an optimal access plan.

For nicknames, table statistics, and index definitions are captured when the nickname is created. After the initial capture, any changes occurring at the remote table are not automatically reflected at the federated data source. Invoking the stored procedure NNSTAT or the Statistics Update facility against the nickname results in updating of the statistics at the federated server; however, the stored procedure does *not* synchronize the nickname's index specifications between the federated server and the remote data source.

At nickname creation, if the remote table did not have statistics collected, then DB2 II uses a default cardinality of 1000 rows.

► Pushdown

For relational data sources in general, it is generally more efficient to push down the processing of a query or query fragment to the remote data source. The SHIP operator in the **db2exfmt** access plan graph identifies the processing that occurs at the remote data source.

The decision to push down a predicate to the remote source depends upon many considerations such as overall cost, server options, column options, and pushdownability of predicates due to federated server and remote data source differences, as discussed in "Pushdown" on page 78.

When certain predicates are not pushed down, operators such as FILTER and GRPBY appear *above* the SHIP operator in the access plan graph indicating that such processing is occurring at the federated server. The absence of complete pushdown should trigger further investigation as to the causes, and may require tuning the DB2 II server options such as PUSHDOWN, CPU_RATIO, IO_RATIO, COMM_RATE, and DB2_MAXIMAL_PUSHDOWN, or nickname options to achieve the desired pushdown.

► Join method

DB2 supports three join methods: Nested loop, merge scan, and hash join. When a federated query joins tables at different data sources, one or more of these join methods is invoked to execute the federated query at the federated server.

In some cases, a join may be executed at the federated server even when the nicknames involved refer to tables at the *same* remote data source. Such occurrences need to be investigated for possible cause and possible resolution to achieve optimal performance.

Depending upon the number of rows involved in the join, and the number and types of predicates involved, a different join method than the one chosen may be appropriate for optimal performance. A thorough understanding of the circumstances that favor a particular join method for superior performance is required to determine whether the join method chosen by the DB2 optimizer needs to be reviewed for relevance. If not, appropriate tuning techniques need to be adopted so that the DB2 optimizer makes the correct join method decision.

► Intra-partition parallelism (INTRA_PARALLEL = NO)

Intra-partition involves dividing a query into multiple concurrent parts that run in parallel by multiple processes on a single database partition. It is enabled when the database manager configuration parameter INTRA_PARALLEL is set to YES, and the degree of parallelism is defined by the DFT_DEGREE database configuration parameter (this can be overridden by the SET CURRENT DEGREE SQL statement).

When the federated query accesses both nickname and local data, and intra-partition parallelism is enabled, access to the local data can occur in parallel while nicknames only run serially. This can allow queries to run faster in an SMP environment.

The Local Table Queue² (LTQ) operator in the **db2exfmt** access plan graph indicates the use of intra-partition parallelism. The Local Merging Table Queue (LMTQ) operator indicates a type of LTQ that merges sorted data from multiple agents.

► Inter-partition parallelism

Inter-partition parallelism involved dividing a single query into multiple parts that run in parallel on different partitions of a partitioned database. It requires the database partition feature (DPF) to be installed.

In queries that reference local and/or remote data sources, DB2 II can distribute nickname data to the partitions for parallel processing. There are no requirements on setting any instance or database configuration parameters to enable inter-partition parallelism; however, only parts of a query that reference fenced wrappers can run in parallel.

The Directed Table Queue (DTQ) and Broadcast Table Queue (BTQ) operator in the **db2exfmt** access path graph indicate the use of inter-partition parallelism. A DTQ sends data to specific partitions based on partition keys or predicates. A BTQ broadcasts rows to several partitions defined by target or destination partitioning. Merging table queues (MDTQ, MBTQ) try to preserve the order of the stream of rows at the receiving side by merging the locally ordered rows from the sending side. A computational partition group (CPG) defines a set of partitions, other than IBMCATNODEGROUP, that can be used

² A table queue is a mechanism to communicate data between db2 agents.

to run the nickname parts of a join query in parallel. This can possibly make the query run faster if the amount of nickname data that participates in the join is large.

The following applies to the server options identified in Table B-5 on page 464:

- ▶ **COMM_RATE**
The default is 2, and the value is in mega-bytes/second.
- ▶ **CPU_RATIO**
A value less than one (that is, 0.00001) means that the data source has more CPU capacity than the federated server.
- ▶ **IO_RATIO**
A value less than one (that is, 0.00001) means that the data source has a faster disk I/O rate than the federated server.
- ▶ **VARCHAR_NO_TRAILING_BLANKS**
This option affects performance with Oracle data sources. With `VARCHAR_NO_TRAILING_BLANKS = 'N'` (default), for VARCHAR and VARCHAR2 columns in joins and filters, DB2 II sends `RPAD(colname)=` to Oracle, and Oracle cannot use an index to process this join or filter, possibly causing poor performance. The option can be set either for an entire server definition, or as a column option on an individual column of a nickname.

Join of nicknames referencing Oracle and SQL server

We used the 32-bit instance for this example. Our objective was to review the access plan in a **db2exfmt** output for a federated query that joined nicknames at two different data sources and no local data access.

Our SQL query ranks customers based on the their placement of large quantity orders. Large quantity orders are defined as those orders whose total quantity is above 300 in our case. The customer table resides on the SQL server, while the orders and line item table reside on Oracle.

Example B-8 shows the complete **db2exfmt** output for our SQL query. The SQL statement we issued has been highlighted under “Original statement” in this output.

Example: B-8 db2exfmt output - Join of nicknames

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

***** EXPLAIN INSTANCE *****

DB2_VERSION: 08.02.0
SOURCE_NAME: SQLC2E05
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-06-24-08.52.14.146220
EXPLAIN_REQUESTER: DB2I32

Database Context:

Parallelism: None
CPU Speed: 4.841528e-07
Comm Speed: 100
Buffer Pool size: 1000
Sort Heap size: 256
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 1020

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 203 -----

QUERYNO: 1
QUERYTAG:
Statement Type: Select
Updatable: No
Deletable: No
Query Degree: 1

Original Statement:

```
SELECT C_NAME, C_CUSTKEY, O_ORDERKEY, O_ORDERDATE, O_TOTALPRICE,
       SUM(L_QUANTITY)
FROM MSS.CUSTOMER, ORA.ORDERS, ORA.LINEITEM
WHERE O_ORDERKEY IN
      (SELECT L_ORDERKEY
       FROM ORA.LINEITEM
       GROUP BY L_ORDERKEY
```

```

HAVING SUM(L_QUANTITY) > 300 ) AND.C_CUSTKEY = O_CUSTKEY AND.O_ORDERKEY =
      L_ORDERKEY
GROUP BY C_NAME, C_CUSTKEY, O_ORDERKEY, O_ORDERDATE, O_TOTALPRICE
ORDER BY O_TOTALPRICE DESC, O_ORDERDATE
FETCH FIRST 100 ROWS ONLY

```

Optimized Statement:

```

-----
SELECT Q9.$C0 AS "C_NAME", Q9.$C1 AS "C_CUSTKEY", Q9.$C2 AS "O_ORDERKEY",
      Q9.$C3 AS "O_ORDERDATE", Q9.$C4 AS "O_TOTALPRICE", Q9.$C5
FROM
  (SELECT Q8.$C0, Q8.$C1, Q8.$C2, Q8.$C3, Q8.$C4, SUM(Q8.$C5)
   FROM
     (SELECT Q7.C_NAME, Q7.C_CUSTKEY, Q6.O_ORDERKEY, Q6.O_ORDERDATE,
            Q6.O_TOTALPRICE, Q5.L_QUANTITY
      FROM ORA.LINEITEM AS Q5, ORA.ORDERS AS Q6, MSS.CUSTOMER AS Q7
      WHERE (Q6.O_ORDERKEY = Q5.L_ORDERKEY) AND (Q7.C_CUSTKEY = Q6.O_CUSTKEY)
            AND Q6.O_ORDERKEY = ANY
        (SELECT Q3.$C1
         FROM
           (SELECT SUM(Q2.$C1), Q2.$C0
            FROM
              (SELECT Q1.L_ORDERKEY, Q1.L_QUANTITY
               FROM ORA.LINEITEM AS Q1) AS Q2
             GROUP BY Q2.$C0) AS Q3
          WHERE (300 < Q3.$C0)) ) AS Q8
      GROUP BY Q8.$C4, Q8.$C3, Q8.$C2, Q8.$C1, Q8.$C0) AS Q9
   ORDER BY Q9.$C4 DESC, Q9.$C3

```

Access Plan:

```

-----
Total Cost: 1.24987e+12
Query Degree:1

```

```

      Rows
      RETURN
      ( 1)
      Cost
      I/O
      |
      100
      GRPBY
      ( 2)
      1.24987e+12
      2.94129e+11
      |
      2.9993e+07
      TBSCAN

```

```

      ( 3)
      1.24987e+12
      2.94129e+11
      |
      2.9993e+07
      SORT
      ( 4)
      1.24986e+12
      2.94128e+11
      |
      2.9993e+07
      MSJOIN
      ( 5)
      1.24984e+12
      2.94127e+11
      /-----\
      1.5e+06      19.9953
      TBSCAN      FILTER
      ( 6)      ( 10)
      583190      1.24984e+12
      65026      2.94127e+11
      |
      1.5e+06      2.9993e+07
      SORT      TBSCAN
      ( 7)      ( 11)
      521190      1.24984e+12
      49719      2.94127e+11
      |
      1.5e+06      2.9993e+07
      SHIP      SORT
      ( 8)      ( 12)
      135200      1.24984e+12
      34412      2.94127e+11
      |
      1.5e+06      2.9993e+07
      NICKNM: MSS SHIP
      CUSTOMER      ( 13)
      1.24983e+12
      2.94126e+11
      /-----\
      1.5e+07      5.99861e+07
      NICKNM: ORA NICKNM: ORA
      ORDERS      LINEITEM

```

1) RETURN: (Return Result)
Cumulative Total Cost: 1.24987e+12

Cumulative CPU Cost: 2.04573e+17
Cumulative I/O Cost: 2.94129e+11
Cumulative Re-Total Cost: 3.78209e+06
Cumulative Re-CPU Cost: 1.1724e+11
Cumulative Re-I/O Cost: 952160
Cumulative First Row Cost: 1.24986e+12
Estimated Bufferpool Buffers: 952160
Remote communication cost:1.71231e+07

Arguments:

BLDLEVEL: (Build level)
DB2 v8.1.1.64 : s040509
ENVVAR : (Environment Variable)
DB2_EXTENDED_OPTIMIZATION = ON
STMTHEAP: (Statement heap size)
8192

Input Streams:

15) From Operator #2

Estimated number of rows: 100
Number of columns: 6
Subquery predicate ID: Not Applicable

Column Names:

+Q10.O_TOTALPRICE(D)+Q10.O_ORDERDATE(A)
+Q10.C_CUSTKEY(A)+Q10.O_ORDERKEY(A)+Q10.\$C5
+Q10.C_NAME

2) GRPBY : (Group By)

Cumulative Total Cost: 1.24987e+12
Cumulative CPU Cost: 2.04573e+17
Cumulative I/O Cost: 2.94129e+11
Cumulative Re-Total Cost: 3.77563e+06
Cumulative Re-CPU Cost: 1.03893e+11
Cumulative Re-I/O Cost: 952160
Cumulative First Row Cost: 1.24986e+12
Estimated Bufferpool Buffers: 952160
Remote communication cost:1.71231e+07

Arguments:

AGGMODE : (Aggregation Mode)
COMPLETE
GROUPBYC: (Group By columns)

```

TRUE
GROUPBYN: (Number of Group By columns)
4
GROUPBYR: (Group By requirement)
1: Q8.C_CUSTKEY
GROUPBYR: (Group By requirement)
2: Q8.O_ORDERKEY
GROUPBYR: (Group By requirement)
3: Q8.O_ORDERDATE
GROUPBYR: (Group By requirement)
4: Q8.O_TOTALPRICE
GROUPBYR: (Group By requirement)
5: Q8.C_NAME
ONEFETCH: (One Fetch flag)
FALSE

```

Input Streams:

14) From Operator #3

```

Estimated number of rows: 2.9993e+07
Number of columns: 6
Subquery predicate ID: Not Applicable

```

Column Names:

```

+Q8.O_TOTALPRICE(D)+Q8.O_ORDERDATE(A)
+Q8.C_CUSTKEY(A)+Q8.O_ORDERKEY(A)
+Q8.L_QUANTITY+Q8.C_NAME

```

Output Streams:

15) To Operator #1

```

Estimated number of rows: 100
Number of columns: 6
Subquery predicate ID: Not Applicable

```

Column Names:

```

+Q10.O_TOTALPRICE(D)+Q10.O_ORDERDATE(A)
+Q10.C_CUSTKEY(A)+Q10.O_ORDERKEY(A)+Q10.$C5
+Q10.C_NAME

```

```

3) TBSCAN: (Table Scan)
Cumulative Total Cost: 1.24987e+12
Cumulative CPU Cost: 2.04573e+17

```


Cumulative I/O Cost: 2.94129e+11
Cumulative Re-Total Cost: 3.772e+06
Cumulative Re-CPU Cost: 9.63947e+10
Cumulative Re-I/O Cost: 952160
Cumulative First Row Cost: 1.24986e+12
Estimated Bufferpool Buffers: 952160
Remote communication cost:1.71231e+07

Arguments:

MAXPAGES: (Maximum pages for prefetch)
ALL
PREFETCH: (Type of Prefetch)
SEQUENTIAL
SCANDIR : (Scan Direction)
FORWARD

Input Streams:

13) From Operator #4

Estimated number of rows: 2.9993e+07
Number of columns: 6
Subquery predicate ID: Not Applicable

Column Names:

+Q8.O_TOTALPRICE(D)+Q8.O_ORDERDATE(A)
+Q8.C_CUSTKEY(A)+Q8.O_ORDERKEY(A)
+Q8.L_QUANTITY+Q8.C_NAME

Output Streams:

14) To Operator #2

Estimated number of rows: 2.9993e+07
Number of columns: 6
Subquery predicate ID: Not Applicable

Column Names:

+Q8.O_TOTALPRICE(D)+Q8.O_ORDERDATE(A)
+Q8.C_CUSTKEY(A)+Q8.O_ORDERKEY(A)
+Q8.L_QUANTITY+Q8.C_NAME

4) SORT : (Sort)
Cumulative Total Cost: 1.24986e+12

Cumulative CPU Cost: 2.04573e+17
Cumulative I/O Cost: 2.94128e+11
Cumulative Re-Total Cost: 0
Cumulative Re-CPU Cost: 0
Cumulative Re-I/O Cost: 952160
Cumulative First Row Cost: 1.24986e+12
Estimated Bufferpool Buffers: 1.23768e+06
Remote communication cost:1.71231e+07

Arguments:

DUPLWARN: (Duplicates Warning flag)
FALSE
NUMROWS : (Estimated number of rows)
29993024
ROWWIDTH: (Estimated width of rows)
60
SORTKEY : (Sort Key column)
1: Q8.O_TOTALPRICE(D)
SORTKEY : (Sort Key column)
2: Q8.O_ORDERDATE(A)
SORTKEY : (Sort Key column)
3: Q8.C_CUSTKEY(A)
SORTKEY : (Sort Key column)
4: Q8.O_ORDERKEY(A)
SPILLED : (Pages spilled to bufferpool or disk)
952160
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

12) From Operator #5

Estimated number of rows: 2.9993e+07
Number of columns: 6
Subquery predicate ID: Not Applicable

Column Names:

+Q8.L_QUANTITY+Q8.O_TOTALPRICE+Q8.O_ORDERDATE
+Q8.O_ORDERKEY+Q8.C_CUSTKEY+Q8.C_NAME

Output Streams:

13) To Operator #3

Estimated number of rows: 2.9993e+07
Number of columns: 6
Subquery predicate ID: Not Applicable

Column Names:

+Q8.O_TOTALPRICE(D)+Q8.O_ORDERDATE(A)
+Q8.C_CUSTKEY(A)+Q8.O_ORDERKEY(A)
+Q8.L_QUANTITY+Q8.C_NAME

5) MSJOIN: (Merge Scan Join)

Cumulative Total Cost: 1.24984e+12
Cumulative CPU Cost: 2.04572e+17
Cumulative I/O Cost: 2.94127e+11
Cumulative Re-Total Cost: 1.24984e+12
Cumulative Re-CPU Cost: 2.04572e+17
Cumulative Re-I/O Cost: 2.94127e+11
Cumulative First Row Cost: 1.24984e+12
Estimated Bufferpool Buffers: 285515
Remote communication cost:1.71231e+07

Arguments:

EARLYOUT: (Early Out flag)
NONE
INNERCOL: (Inner Order Columns)
1: Q6.O_CUSTKEY(A)
OUTERCOL: (Outer Order columns)
1: Q7.C_CUSTKEY(A)
TEMPSIZE: (Temporary Table Page Size)
4096

Predicates:

8) Predicate used in Join
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 6.66667e-07

Predicate Text:

(Q7.C_CUSTKEY = Q6.O_CUSTKEY)

Input Streams:

4) From Operator #6

Estimated number of rows: 1.5e+06
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q7.C_CUSTKEY(A)+Q7.C_NAME

11) From Operator #10

Estimated number of rows: 19.9953
Number of columns: 5
Subquery predicate ID: Not Applicable

Column Names:

+Q6.O_CUSTKEY(A)+Q5.L_QUANTITY+Q6.O_TOTALPRICE
+Q6.O_ORDERDATE+Q6.O_ORDERKEY

Output Streams:

12) To Operator #4

Estimated number of rows: 2.9993e+07
Number of columns: 6
Subquery predicate ID: Not Applicable

Column Names:

+Q8.L_QUANTITY+Q8.O_TOTALPRICE+Q8.O_ORDERDATE
+Q8.O_ORDERKEY+Q8.C_CUSTKEY+Q8.C_NAME

6) TBSCAN: (Table Scan)

Cumulative Total Cost: 583190
Cumulative CPU Cost: 1.23151e+10
Cumulative I/O Cost: 65026
Cumulative Re-Total Cost: 61999.6
Cumulative Re-CPU Cost: 4.32908e+09
Cumulative Re-I/O Cost: 15307
Cumulative First Row Cost: 523276
Estimated Bufferpool Buffers: 15307
Remote communication cost:824343

Arguments:

JN INPUT: (Join input leg)

OUTER
MAXPAGES: (Maximum pages for prefetch)
ALL
PREFETCH: (Type of Prefetch)
SEQUENTIAL
SCANDIR : (Scan Direction)
FORWARD

Input Streams:

3) From Operator #7

Estimated number of rows: 1.5e+06
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q7.C_CUSTKEY(A)+Q7.C_NAME

Output Streams:

4) To Operator #5

Estimated number of rows: 1.5e+06
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q7.C_CUSTKEY(A)+Q7.C_NAME

7) SORT : (Sort)
Cumulative Total Cost: 521190
Cumulative CPU Cost: 7.98603e+09
Cumulative I/O Cost: 49719
Cumulative Re-Total Cost: 0
Cumulative Re-CPU Cost: 0
Cumulative Re-I/O Cost: 15307
Cumulative First Row Cost: 521190
Estimated Bufferpool Buffers: 49719
Remote communication cost:824343

Arguments:

DUPLWARN: (Duplicates Warning flag)
FALSE

NUMROWS : (Estimated number of rows)
 1500000
 ROWWIDTH: (Estimated width of rows)
 36
 SORTKEY : (Sort Key column)
 1: Q7.C_CUSTKEY(A)
 SPILLED : (Pages spilled to bufferpool or disk)
 15307
 TEMPSIZE: (Temporary Table Page Size)
 4096
 UNIQUE : (Uniqueness required flag)
 FALSE

Input Streams:

2) From Operator #8

Estimated number of rows: 1.5e+06
 Number of columns: 2
 Subquery predicate ID: Not Applicable

Column Names:

+Q7.C_NAME+Q7.C_CUSTKEY

Output Streams:

3) To Operator #6

Estimated number of rows: 1.5e+06
 Number of columns: 2
 Subquery predicate ID: Not Applicable

Column Names:

+Q7.C_CUSTKEY(A)+Q7.C_NAME

8) SHIP : (Ship)

Cumulative Total Cost: 135200
 Cumulative CPU Cost: 1.1371e+09
 Cumulative I/O Cost: 34412
 Cumulative Re-Total Cost: 135200
 Cumulative Re-CPU Cost: 1.1371e+09
 Cumulative Re-I/O Cost: 34412
 Cumulative First Row Cost: 25.0071
 Estimated Bufferpool Buffers: 34412
 Remote communication cost:824343

Arguments:

```
-----  
CSERQY  : (Remote common subexpression)  
        FALSE  
DSTSEVER: (Destination (ship to) server)  
        - (NULL).  
RMTQTXT : (Remote statement)  
        SELECT AO."C_CUSTKEY", AO."C_NAME" FROM "TPCD"."CUSTOMER" AO  
SRCSEVER: (Source (ship from) server)  
        SQLSERV  
STREAM  : (Remote stream)  
        FALSE
```

Input Streams:

```
-----  
1) From Object MSS.CUSTOMER
```

```
Estimated number of rows: 1.5e+06  
Number of columns: 3  
Subquery predicate ID: Not Applicable
```

Column Names:

```
-----  
+Q7.$RID$+Q7.C_NAME+Q7.C_CUSTKEY
```

Output Streams:

```
-----  
2) To Operator #7
```

```
Estimated number of rows: 1.5e+06  
Number of columns: 2  
Subquery predicate ID: Not Applicable
```

Column Names:

```
-----  
+Q7.C_NAME+Q7.C_CUSTKEY
```

10) FILTER: (Filter)

```
Cumulative Total Cost: 1.24984e+12  
Cumulative CPU Cost: 2.04572e+17  
Cumulative I/O Cost: 2.94127e+11  
Cumulative Re-Total Cost: 1.10107e+06  
Cumulative Re-CPU Cost: 9.06373e+10  
Cumulative Re-I/O Cost: 270208  
Cumulative First Row Cost: 1.24984e+12  
Estimated Bufferpool Buffers: 270208
```

Remote communication cost:1.62988e+07

Arguments:

JN INPUT: (Join input leg)
INNER

Predicates:

8) Residual Predicate
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 6.66667e-07

Predicate Text:

(Q7.C_CUSTKEY = Q6.O_CUSTKEY)

Input Streams:

10) From Operator #11

Estimated number of rows: 2.9993e+07
Number of columns: 5
Subquery predicate ID: Not Applicable

Column Names:

+Q6.O_CUSTKEY(A)+Q5.L_QUANTITY+Q6.O_TOTALPRICE
+Q6.O_ORDERDATE+Q6.O_ORDERKEY

Output Streams:

11) To Operator #5

Estimated number of rows: 19.9953
Number of columns: 5
Subquery predicate ID: Not Applicable

Column Names:

+Q6.O_CUSTKEY(A)+Q5.L_QUANTITY+Q6.O_TOTALPRICE
+Q6.O_ORDERDATE+Q6.O_ORDERKEY

11) TBSCAN: (Table Scan)

Cumulative Total Cost: 1.24984e+12

Cumulative CPU Cost: 2.04572e+17
Cumulative I/O Cost: 2.94127e+11
Cumulative Re-Total Cost: 1.10107e+06
Cumulative Re-CPU Cost: 9.06373e+10
Cumulative Re-I/O Cost: 270208
Cumulative First Row Cost: 1.24984e+12
Estimated Bufferpool Buffers: 270208
Remote communication cost:1.62988e+07

Arguments:

MAXPAGES: (Maximum pages for prefetch)
ALL
PREFETCH: (Type of Prefetch)
SEQUENTIAL
SCANDIR : (Scan Direction)
FORWARD

Input Streams:

9) From Operator #12

Estimated number of rows: 2.9993e+07
Number of columns: 5
Subquery predicate ID: Not Applicable

Column Names:

+Q6.0_CUSTKEY(A)+Q5.L_QUANTITY+Q6.0_TOTALPRICE
+Q6.0_ORDERDATE+Q6.0_ORDERKEY

Output Streams:

10) To Operator #10

Estimated number of rows: 2.9993e+07
Number of columns: 5
Subquery predicate ID: Not Applicable

Column Names:

+Q6.0_CUSTKEY(A)+Q5.L_QUANTITY+Q6.0_TOTALPRICE
+Q6.0_ORDERDATE+Q6.0_ORDERKEY

12) SORT : (Sort)

Cumulative Total Cost: 1.24984e+12
Cumulative CPU Cost: 2.04572e+17

Cumulative I/O Cost: 2.94127e+11
Cumulative Re-Total Cost: 0
Cumulative Re-CPU Cost: 0
Cumulative Re-I/O Cost: 270208
Cumulative First Row Cost: 1.24984e+12
Estimated Bufferpool Buffers: 1.48701e+06
Remote communication cost:1.62988e+07

Arguments:

DUPLWARN: (Duplicates Warning flag)
FALSE
NUMROWS : (Estimated number of rows)
29993026
ROWWIDTH: (Estimated width of rows)
32
SORTKEY : (Sort Key column)
1: Q6.0_CUSTKEY(A)
SPILLED : (Pages spilled to bufferpool or disk)
270208
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

8) From Operator #13

Estimated number of rows: 2.9993e+07
Number of columns: 5
Subquery predicate ID: Not Applicable

Column Names:

+Q5.L_QUANTITY+Q6.0_TOTALPRICE+Q6.0_ORDERDATE
+Q6.0_CUSTKEY+Q6.0_ORDERKEY

Output Streams:

9) To Operator #11

Estimated number of rows: 2.9993e+07
Number of columns: 5
Subquery predicate ID: Not Applicable

Column Names:

```
+Q6.O_CUSTKEY(A)+Q5.L_QUANTITY+Q6.O_TOTALPRICE
+Q6.O_ORDERDATE+Q6.O_ORDERKEY
```

13) SHIP : (Ship)

```
Cumulative Total Cost: 1.24983e+12
Cumulative CPU Cost: 2.04572e+17
Cumulative I/O Cost: 2.94126e+11
Cumulative Re-Total Cost: 1.24983e+12
Cumulative Re-CPU Cost: 2.04572e+17
Cumulative Re-I/O Cost: 2.94126e+11
Cumulative First Row Cost: 3.72551e+07
Estimated Bufferpool Buffers: 1.2168e+06
Remote communication cost:1.62988e+07
```

Arguments:

```
CSERQY : (Remote common subexpression)
FALSE
```

```
DSTSEVER: (Destination (ship to) server)
- (NULL).
```

```
RMTQTX : (Remote statement)
```

```
SELECT A1."O_ORDERKEY", A1."O_CUSTKEY", A1."O_ORDERDATE",
A1."O_TOTALPRICE", A0."L_QUANTITY" FROM "IITEST"."LINEITEM" A0,
"IITEST"."ORDERS" A1 WHERE (A1."O_ORDERKEY"= ANY (SELECT A2."L_ORDERKEY" FROM
"IITEST"."LINEITEM" A2 GROUP BY A2."L_ORDERKEY" HAVING (300 < SUM(
A2."L_QUANTITY")))) AND (A1."O_ORDERKEY" = A0."L_ORDERKEY")
```

```
SRCSEVER: (Source (ship from) server)
```

```
ORASERV
```

```
STREAM : (Remote stream)
FALSE
```

Input Streams:

5) From Object ORA.LINEITEM

```
Estimated number of rows: 5.99861e+07
Number of columns: 3
Subquery predicate ID: Not Applicable
```

Column Names:

```
+Q5.$RID$+Q5.L_QUANTITY+Q5.L_ORDERKEY
```

6) From Object ORA.ORDERS

```
Estimated number of rows: 1.5e+07
Number of columns: 2
Subquery predicate ID: Not Applicable
```

Column Names:

+Q6.0_TOTALPRICE+Q6.0_CUSTKEY

7) From Object ORA.LINEITEM

Estimated number of rows: 5.99861e+07

Number of columns: 3

Subquery predicate ID: 9

Column Names:

+Q1.\$RID\$+Q1.L_QUANTITY+Q1.L_ORDERKEY

Output Streams:

8) To Operator #12

Estimated number of rows: 2.9993e+07

Number of columns: 5

Subquery predicate ID: Not Applicable

Column Names:

+Q5.L_QUANTITY+Q6.0_TOTALPRICE+Q6.0_ORDERDATE

+Q6.0_CUSTKEY+Q6.0_ORDERKEY

Objects Used in Access Plan:

Schema: MSS

Name: CUSTOMER

Type: Nickname

Time of creation: 2004-06-10-22.24.53.208724

Last statistics update:

Number of columns: 8

Number of rows: 1500000

Width of rows: 43

Number of buffer pool pages: 34412

Distinct row values: No

Tablespace name:

Tablespace overhead: 24.100000

Tablespace transfer rate: 0.900000

Source for statistics: Single Node

Prefetch page count: 32

Container extent page count: 32

Schema: ORA
Name: LINEITEM
Type: Nickname
Time of creation: 2004-06-10-08.38.39.249747
Last statistics update: 2004-06-10-19.56.51.065544
Number of columns: 16
Number of rows: 59986052
Width of rows: 41
Number of buffer pool pages: 2081039
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: ORA
Name: ORDERS
Type: Nickname
Time of creation: 2004-06-10-08.38.39.158115
Last statistics update: 2004-06-10-19.56.50.452787
Number of columns: 9
Number of rows: 15000000
Width of rows: 49
Number of buffer pool pages: 443840
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Access plan description

The access plan graph for our SQL query has been highlighted in Example B-8 on page 469 under “Access Plan”.

Reading this graph bottom up reveals the following:

- The bottom left part of the graph indicates that the CUSTOMER nickname has 1.5 million rows according to the DB2 II catalog statistics, and that all these rows are processed by the SHIP operator (8). However, only three columns (see the Input Streams for the SHIP operator 8) were retrieved, including the RID, C_CNAME, and C_CUSTKEY.

- ▶ The rows from the SHIP operator are input to the SORT operator (7) with two columns (C_NAME and C_CUSTKEY) being passed (see the Output Streams for the SHIP operator 8, which corresponds to the Input Streams for the SORT operator 7). These rows are sorted ascending on the C_CUSTKEY column (see Arguments in the SORT operator 7) and stored in a temporary table.
- ▶ The sorted rows are then scanned via a table scan (TBSCAN operator 6) and used as the outer table, as indicated by OUTERCOL in the Arguments of the merge scan join (MSJOIN operator 5). The inner table of this merge scan join is estimated to have 19.9953 rows, and the result of the merge scan join is estimated to be 2.9993e+07 rows.
- ▶ The 19.9953 rows for the inner table of the join are formed from a series of executions involving a SHIP (operator 13), SORT (operator 12), TBSCAN (operator 11), and FILTER (operator 10).
 - The SHIP operator 13 estimates 2.9993e+07 rows (5 columns) are retrieved from the Oracle data source—the RMTQTXT field in Arguments for the SHIP operator 8 contains the text of the SQL statement sent to the Oracle data source as being a join of nicknames ORDERS (1.5e+07 cardinality according to the DB2 II catalog) and LINEITEM (5.99861e+07 cardinality according to the DB2 II catalog).
 - The SORT operator 12 sorts the rows in O-CUSTKEY ascending order and writes them to a temporary table.
 - The TBSCAN operator 11 retrieves the sorted rows in the temporary table and passes them to the FILTER operator 10.
 - The FILTER operator 10 applies the residual predicate (Q7.C_CUSTKEY = Q6.O_CUSTKEY), estimating that 19.9953 rows will qualify.
- ▶ The estimated 2.9993e+07 rows (6 columns) result of the merge scan join (MSJOIN operator 5) is input to SORT operator 4, which sorts on four columns (descending O_TOTALPRICE, ascending O_ORDERDATE, ascending C_CUSTKEY, ascending O_ORDERKEY) and written to a temporary table.
- ▶ The TBSCAN operator 3 scans this temporary table and passes all the 2.9993e+07 rows to the GRPBY operator 2.
- ▶ The GRPBY operator 2 groups on five columns and estimates that 100 rows will be returned to the user via the RETURN operator 1.
- ▶ The total cost is estimated to be 1.24987e+12 timerons, and there is no parallelism involved (Query Degree:1 and no table queue operators in the Access Plan).

Analysis

Important: Bearing in mind that the EXPLAIN output information is DB2 optimizer estimates only and does not necessarily reflect actual runtime metrics, you should be cautious in drawing conclusions about relative costs associated with each operator, as well as relying on the number of estimated rows retrieved or returned by the various operators. EXPLAIN output information is best used in conjunction with runtime metrics gathered through monitoring.

The following observations apply to the **db2exfmt** output shown in Example B-8 on page 469.

- ▶ There is only one SHIP operator to the Oracle data source, and it references two nicknames (ORA.ORDERS and ORA.LINEITEM). This indicates that the join of the ORDERS and LINEITEM nicknames is being pushed down to Oracle, which tends to be a positive indicator.
- ▶ The cardinality of the nicknames appear to be valid since the DB2 optimizer default of 1000 rows does not appear. This, however, does not guarantee that the DB2 II catalog reflects the actual number of rows at the remote data source.
- ▶ There are a few sorts involved (operators 4, 7 and 12) with a large numbers of rows involved. The Database Context shows the Sort Heap size to be 256 pages, while the Buffer Pool size is 1000. This causes spillover to buffer pool and disk (temporary table space), which can impact performance. SORT operator 7 estimates a need for 49719 buffers, while SORT operator 12 estimates a need for 14,870,100 buffers. While this indicates a potential performance improvement to be gained by tuning the sort heap and number of buffers, it appears that a significant portion of the total estimated cost of the query (1.24987e+12) comes from the SHIP operator to Oracle that is expected to return about 29,000,000 rows.

Our analysis points to satisfactory pushdown of predicates, and possible tuning of the sort heap and number of buffers after actual runtime metrics are gathered for this query.

INTRA_PARALLEL = YES (intra-partition enabled)

We used the 32-bit instance for this example. Our objective was to review the access plan in a **db2exfmt** output for a federated query that joined nickname data at a remote data source with local non-DPF data in an SMP environment with the database manager configuration option INTRA_PARALLEL = YES (intra-partition parallelism enabled).

We used the same query that was used in “Join of nicknames referencing Oracle and SQL server” on page 469, with the difference being that the orders and line item table now reside on local DB2 UDB non-DPF tables using schema name TPCD, while the customer table continues to reside on the SQL server.

Example B-9 shows the complete **db2exfmt** output for our SQL query. The SQL statement we issued has been highlighted under “Original statement” in this output.

Example: B-9 db2exfmt output for intra-partition parallelism enabled

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

***** EXPLAIN INSTANCE *****

DB2_VERSION: 08.02.0
SOURCE_NAME: SQLC2E05
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-06-24-17.35.31.278653
EXPLAIN_REQUESTER: DB2I32

Database Context:

Parallelism: Intra-Partition Parallelism
CPU Speed: 4.841528e-07
Comm Speed: 100
Buffer Pool size: 1000
Sort Heap size: 256
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 1020

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 203 -----

QUERYNO: 1
 QUERYTAG:
 Statement Type: Select
 Updatable: No
 Deletable: No
 Query Degree: -1

Original Statement:

```

-----
SELECT C_NAME, C_CUSTKEY, O_ORDERKEY, O_ORDERDATE, O_TOTALPRICE,
       SUM(L_QUANTITY)
FROM mss.CUSTOMER, TPCD.ORDERS, TPCD.LINEITEM
WHERE O_ORDERKEY IN
      (SELECT L_ORDERKEY
       FROM TPCD.LINEITEM
       GROUP BY L_ORDERKEY
       HAVING SUM(L_QUANTITY) > 300 ) AND.C_CUSTKEY = O_CUSTKEY AND.O_ORDERKEY =
      L_ORDERKEY
GROUP BY C_NAME, C_CUSTKEY, O_ORDERKEY, O_ORDERDATE, O_TOTALPRICE
ORDER BY O_TOTALPRICE DESC, O_ORDERDATE
FETCH FIRST 100 ROWS ONLY
  
```

Optimized Statement:

```

-----
SELECT Q7.$C5 AS "C_NAME", Q7.$C4 AS "C_CUSTKEY", Q7.$C3 AS "O_ORDERKEY",
       Q7.$C2 AS "O_ORDERDATE", Q7.$C1 AS "O_TOTALPRICE", Q7.$C0
FROM
  (SELECT SUM(Q6.$C5), Q6.$C0, Q6.$C1, Q6.$C2, Q6.$C3, Q6.$C4
   FROM
     (SELECT Q4.O_TOTALPRICE, Q4.O_ORDERDATE, Q4.O_ORDERKEY, Q5.C_CUSTKEY,
      Q5.C_NAME, Q3.$C0
     FROM
       (SELECT SUM(Q2.$C1), Q2.$C0
        FROM
          (SELECT Q1.L_ORDERKEY, Q1.L_QUANTITY
           FROM TPCD.LINEITEM AS Q1) AS Q2
          GROUP BY Q2.$C0) AS Q3, TPCD.ORDERS AS Q4, MSS.CUSTOMER AS Q5
          WHERE (Q3.$C1 = Q4.O_ORDERKEY) AND (+3.000000000000000E+002 < Q3.$C0)
            AND (Q5.C_CUSTKEY = Q4.O_CUSTKEY)) AS Q6
          GROUP BY Q6.$C4, Q6.$C3, Q6.$C2, Q6.$C1, Q6.$C0) AS Q7
      ORDER BY Q7.$C1 DESC, Q7.$C2
  
```

Access Plan:

 Total Cost: 1.30554e+07
 Query Degree:8

Rows

```

RETURN
( 1)
Cost
I/O
|
100
GRPBY
( 2)
1.30554e+07
1.91732e+06
|
954.234
NLJOIN
( 3)
1.30554e+07
1.91732e+06
/-----+-----\
1272.23      0.750047
LMTQ          SHIP
( 4)          ( 15)
1.2946e+07    68.7753
1.91295e+06    2.75005
|              |
1272.23      1.5e+06
TBSCAN      NICKNM: MSS
( 5)        CUSTOMER
1.2946e+07
1.91295e+06
|
1272.23
SORT
( 6)
1.2946e+07
1.91295e+06
|
1272.23
NLJOIN
( 7)
1.2946e+07
1.91295e+06
/-----+-----\
1272.23      1
FILTER        FETCH
( 8)          ( 13)
1.27556e+07    100.034
1.90533e+06      4
|              /-----+-----\
3e+07          1      3e+07
GRPBY          IXSCAN  TABLE: TPCD

```

(9)	(14)	ORDERS
1.27426e+07	75.0263	
1.90533e+06	3	
3e+07	3e+07	
TBSCAN	INDEX: TPCD	
(10)	0_OK_OD_OP	
1.2739e+07		
1.90533e+06		
3e+07		
SORT		
(11)		
1.17875e+07		
1.67277e+06		
1.1997e+08		
IXSCAN		
(12)		
5.75522e+06		
1.44021e+06		
1.1997e+08		
INDEX: TPCD		
L_PKSKOKEPDSQN		

1) RETURN: (Return Result)

Cumulative Total Cost: 1.30554e+07

Cumulative CPU Cost: 8.19229e+11

Cumulative I/O Cost: 1.91732e+06

Cumulative Re-Total Cost: 1.07132e+06

Cumulative Re-CPU Cost: 1.20207e+11

Cumulative Re-I/O Cost: 1590.1

Cumulative First Row Cost: 1.29461e+07

Estimated Bufferpool Buffers: 3498.7

Remote communication cost:6881.96

Arguments:

BLDLEVEL: (Build level)

DB2 v8.1.1.64 : s040509

ENVVAR : (Environment Variable)

DB2_EXTENDED_OPTIMIZATION = ON

STMHEAP: (Statement heap size)

8192

Input Streams:

18) From Operator #2

Estimated number of rows: 100

Number of columns: 6

Subquery predicate ID: Not Applicable

Column Names:

+Q8.O_TOTALPRICE(D)+Q8.O_ORDERDATE(A)

+Q8.O_ORDERKEY(A)+Q8.\$C5+Q8.C_CUSTKEY+Q8.C_NAME

2) GRPBY : (Group By)

Cumulative Total Cost: 1.30554e+07

Cumulative CPU Cost: 8.19229e+11

Cumulative I/O Cost: 1.91732e+06

Cumulative Re-Total Cost: 1.07132e+06

Cumulative Re-CPU Cost: 1.20206e+11

Cumulative Re-I/O Cost: 1590.1

Cumulative First Row Cost: 1.29461e+07

Estimated Bufferpool Buffers: 3498.7

Remote communication cost:6881.96

Arguments:

AGGMODE : (Aggregation Mode)

COMPLETE

GROUPBYC: (Group By columns)

TRUE

GROUPBYN: (Number of Group By columns)

1

GROUPBYR: (Group By requirement)

1: Q6.O_ORDERKEY

GROUPBYR: (Group By requirement)

2: Q6.O_ORDERDATE

GROUPBYR: (Group By requirement)

3: Q6.O_TOTALPRICE

GROUPBYR: (Group By requirement)

4: Q6.C_CUSTKEY

GROUPBYR: (Group By requirement)

5: Q6.C_NAME

ONEFETCH: (One Fetch flag)

FALSE

Input Streams:

17) From Operator #3

Estimated number of rows: 954.234
Number of columns: 6
Subquery predicate ID: Not Applicable

Column Names:

+Q6.0_TOTALPRICE(D)+Q6.0_ORDERDATE(A)
+Q6.0_ORDERKEY(A)+Q6.\$C5+Q6.C_NAME+Q6.C_CUSTKEY

Output Streams:

18) To Operator #1

Estimated number of rows: 100
Number of columns: 6
Subquery predicate ID: Not Applicable

Column Names:

+Q8.0_TOTALPRICE(D)+Q8.0_ORDERDATE(A)
+Q8.0_ORDERKEY(A)+Q8.\$C5+Q8.C_CUSTKEY+Q8.C_NAME

3) NLJOIN: (Nested Loop Join)

Cumulative Total Cost: 1.30554e+07
Cumulative CPU Cost: 8.19229e+11
Cumulative I/O Cost: 1.91732e+06
Cumulative Re-Total Cost: 1.07132e+06
Cumulative Re-CPU Cost: 1.20206e+11
Cumulative Re-I/O Cost: 1590.1
Cumulative First Row Cost: 1.29461e+07
Estimated Bufferpool Buffers: 3498.7
Remote communication cost:6881.96

Arguments:

EARLYOUT: (Early Out flag)
LEFT
FETCHMAX: (Override for FETCH MAXPAGES)
IGNORE
ISCANMAX: (Override for ISCAN MAXPAGES)
IGNORE

Predicates:

9) Predicate used in Join
Relational Operator: Equal (=)

Subquery Input Required: No
Filter Factor: 5.00031e-07

Predicate Text:

(Q5.C_CUSTKEY = Q4.O_CUSTKEY)

Input Streams:

14) From Operator #4

Estimated number of rows: 1272.23
Number of columns: 5
Subquery predicate ID: Not Applicable

Column Names:

+Q4.O_TOTALPRICE(D)+Q4.O_ORDERDATE(A)
+Q4.O_ORDERKEY(A)+Q3.\$C0+Q4.O_CUSTKEY

16) From Operator #15

Estimated number of rows: 0.750047
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q5.C_NAME+Q5.C_CUSTKEY

Output Streams:

17) To Operator #2

Estimated number of rows: 954.234
Number of columns: 6
Subquery predicate ID: Not Applicable

Column Names:

+Q6.O_TOTALPRICE(D)+Q6.O_ORDERDATE(A)
+Q6.O_ORDERKEY(A)+Q6.\$C5+Q6.C_NAME+Q6.C_CUSTKEY

4) TQ : (Table Queue)
Cumulative Total Cost: 1.2946e+07
Cumulative CPU Cost: 8.19176e+11

Cumulative I/O Cost: 1.91295e+06
Cumulative Re-Total Cost: 1.03154e+06
Cumulative Re-CPU Cost: 1.20163e+11
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 1.2946e+07
Estimated Bufferpool Buffers: 0

Arguments:

JN INPUT: (Join input leg)
 OUTER
LISTENER: (Listener Table Queue type)
 FALSE
SORTKEY : (Sort Key column)
 1: Q4.0_TOTALPRICE(D)
SORTKEY : (Sort Key column)
 2: Q4.0_ORDERDATE(A)
SORTKEY : (Sort Key column)
 3: Q4.0_ORDERKEY(A)
TQ TYPE : (Table queue type)
 LOCAL
TQDEGREE: (Degree of Intra-Partition parallelism)
 8
TQMERGE : (Merging Table Queue flag)
 TRUE
TQREAD : (Table Queue Read type)
 READ AHEAD
UNIQUE : (Uniqueness required flag)
 FALSE

Input Streams:

13) From Operator #5

Estimated number of rows: 1272.23
Number of columns: 5
Subquery predicate ID: Not Applicable

Column Names:

+Q4.0_TOTALPRICE(D)+Q4.0_ORDERDATE(A)
+Q4.0_ORDERKEY(A)+Q3.\$C0+Q4.0_CUSTKEY

Output Streams:

14) To Operator #3

Estimated number of rows: 1272.23

Number of columns: 5
Subquery predicate ID: Not Applicable

Column Names:

+Q4.0_TOTALPRICE(D)+Q4.0_ORDERDATE(A)
+Q4.0_ORDERKEY(A)+Q3.\$C0+Q4.0_CUSTKEY

5) TBSCAN: (Table Scan)

Cumulative Total Cost: 1.2946e+07
Cumulative CPU Cost: 8.19174e+11
Cumulative I/O Cost: 1.91295e+06
Cumulative Re-Total Cost: 1.03154e+06
Cumulative Re-CPU Cost: 1.20163e+11
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 1.2946e+07
Estimated Bufferpool Buffers: 0

Arguments:

MAXPAGES: (Maximum pages for prefetch)

ALL

PREFETCH: (Type of Prefetch)

NONE

SCANDIR : (Scan Direction)

FORWARD

Input Streams:

12) From Operator #6

Estimated number of rows: 1272.23
Number of columns: 5
Subquery predicate ID: Not Applicable

Column Names:

+Q4.0_TOTALPRICE(D)+Q4.0_ORDERDATE(A)
+Q4.0_ORDERKEY(A)+Q3.\$C0+Q4.0_CUSTKEY

Output Streams:

13) To Operator #4

Estimated number of rows: 1272.23
Number of columns: 5
Subquery predicate ID: Not Applicable


```

Column Names:
-----
+Q4.0_TOTALPRICE(D)+Q4.0_ORDERDATE(A)
+Q4.0_ORDERKEY(A)+Q3.$C0+Q4.0_CUSTKEY

```

```

6) SORT : (Sort)
Cumulative Total Cost: 1.2946e+07
Cumulative CPU Cost: 8.19172e+11
Cumulative I/O Cost: 1.91295e+06
Cumulative Re-Total Cost: 1.03154e+06
Cumulative Re-CPU Cost: 1.20161e+11
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 1.2946e+07
Estimated Bufferpool Buffers: 237648

```

```

Arguments:
-----
DUPLWARN: (Duplicates Warning flag)
FALSE
NUMROWS : (Estimated number of rows)
1273
ROWWIDTH: (Estimated width of rows)
32
SORTKEY : (Sort Key column)
1: Q4.0_TOTALPRICE(D)
SORTKEY : (Sort Key column)
2: Q4.0_ORDERDATE(A)
SORTKEY : (Sort Key column)
3: Q4.0_ORDERKEY(A)
SORTTYPE: (Intra-Partition parallelism sort type)
ROUND ROBIN
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
FALSE

```

```

Input Streams:
-----

```

```

11) From Operator #7

```

```

Estimated number of rows: 1272.23
Number of columns: 6
Subquery predicate ID: Not Applicable

```

```

Column Names:
-----
+Q3.L_ORDERKEY(A)+Q3.$C0+Q4.0_ORDERDATE

```

+Q4.0_TOTALPRICE+Q4.0_ORDERKEY+Q4.0_CUSTKEY

Output Streams:

12) To Operator #5

Estimated number of rows: 1272.23

Number of columns: 5

Subquery predicate ID: Not Applicable

Column Names:

+Q4.0_TOTALPRICE(D)+Q4.0_ORDERDATE(A)

+Q4.0_ORDERKEY(A)+Q3.\$C0+Q4.0_CUSTKEY

7) NLJOIN: (Nested Loop Join)

Cumulative Total Cost: 1.2946e+07

Cumulative CPU Cost: 8.19168e+11

Cumulative I/O Cost: 1.91295e+06

Cumulative Re-Total Cost: 1.03154e+06

Cumulative Re-CPU Cost: 1.20161e+11

Cumulative Re-I/O Cost: 235098

Cumulative First Row Cost: 1.18291e+07

Estimated Bufferpool Buffers: 237648

Arguments:

EARLYOUT: (Early Out flag)

LEFT

FETCHMAX: (Override for FETCH MAXPAGES)

IGNORE

ISCANMAX: (Override for ISCAN MAXPAGES)

IGNORE

Predicates:

7) Predicate used in Join

Relational Operator: Equal (=)

Subquery Input Required: No

Filter Factor: 3.33333e-08

Predicate Text:

(Q3.\$C1 = Q4.0_ORDERKEY)

Input Streams:

6) From Operator #8

Estimated number of rows: 1272.23
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q3.L_ORDERKEY(A)+Q3.\$C0

10) From Operator #13

Estimated number of rows: 1
Number of columns: 4
Subquery predicate ID: Not Applicable

Column Names:

+Q4.0_ORDERKEY(A)+Q4.0_ORDERDATE(A)

+Q4.0_TOTALPRICE+Q4.0_CUSTKEY

Output Streams:

11) To Operator #6

Estimated number of rows: 1272.23
Number of columns: 6
Subquery predicate ID: Not Applicable

Column Names:

+Q3.L_ORDERKEY(A)+Q3.\$C0+Q4.0_ORDERDATE

+Q4.0_TOTALPRICE+Q4.0_ORDERKEY+Q4.0_CUSTKEY

8) FILTER: (Filter)

Cumulative Total Cost: 1.27556e+07
Cumulative CPU Cost: 8.19096e+11
Cumulative I/O Cost: 1.90533e+06
Cumulative Re-Total Cost: 968040
Cumulative Re-CPU Cost: 1.20106e+11
Cumulative Re-I/O Cost: 232559
Cumulative First Row Cost: 1.1829e+07
Estimated Bufferpool Buffers: 232559

Arguments:

JN INPUT: (Join input leg)
OUTER

Predicates:

8) Residual Predicate
Relational Operator: Less Than (<)
Subquery Input Required: No
Filter Factor: 4.24078e-05

Predicate Text:

(+3.000000000000000E+002 < Q3.\$C0)

Input Streams:

5) From Operator #9

Estimated number of rows: 3e+07
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q3.L_ORDERKEY(A)+Q3.\$C0

Output Streams:

6) To Operator #7

Estimated number of rows: 1272.23
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q3.L_ORDERKEY(A)+Q3.\$C0

9) GRPBY : (Group By)
Cumulative Total Cost: 1.27426e+07
Cumulative CPU Cost: 7.92396e+11
Cumulative I/O Cost: 1.90533e+06
Cumulative Re-Total Cost: 955113
Cumulative Re-CPU Cost: 9.34061e+10
Cumulative Re-I/O Cost: 232559
Cumulative First Row Cost: 1.18286e+07

Estimated Bufferpool Buffers: 232559

Arguments:

AGGMODE : (Aggregation Mode)
 FINAL
GROUPBYC: (Group By columns)
 TRUE
GROUPBYN: (Number of Group By columns)
 1
GROUPBYR: (Group By requirement)
 1: Q2.L_ORDERKEY
ONEFETCH: (One Fetch flag)
 FALSE

Input Streams:

4) From Operator #10

 Estimated number of rows: 3e+07
 Number of columns: 2
 Subquery predicate ID: Not Applicable

 Column Names:

 +Q2.L_ORDERKEY(A)+Q2.L_QUANTITY

Output Streams:

5) To Operator #8

 Estimated number of rows: 3e+07
 Number of columns: 2
 Subquery predicate ID: Not Applicable

 Column Names:

 +Q3.L_ORDERKEY(A)+Q3.\$C0

10) TBSCAN: (Table Scan)
 Cumulative Total Cost: 1.2739e+07
 Cumulative CPU Cost: 7.84896e+11
 Cumulative I/O Cost: 1.90533e+06
 Cumulative Re-Total Cost: 951482
 Cumulative Re-CPU Cost: 8.59061e+10
 Cumulative Re-I/O Cost: 232559
 Cumulative First Row Cost: 1.18286e+07

Estimated Bufferpool Buffers: 232559

Arguments:

MAXPAGES: (Maximum pages for prefetch)

ALL

PREFETCH: (Type of Prefetch)

SEQUENTIAL

SCANDIR : (Scan Direction)

FORWARD

Input Streams:

3) From Operator #11

Estimated number of rows: 3e+07

Number of columns: 2

Subquery predicate ID: Not Applicable

Column Names:

+Q2.L_ORDERKEY(A)+Q2.L_QUANTITY

Output Streams:

4) To Operator #9

Estimated number of rows: 3e+07

Number of columns: 2

Subquery predicate ID: Not Applicable

Column Names:

+Q2.L_ORDERKEY(A)+Q2.L_QUANTITY

11) SORT : (Sort)

Cumulative Total Cost: 1.17875e+07

Cumulative CPU Cost: 6.98989e+11

Cumulative I/O Cost: 1.67277e+06

Cumulative Re-Total Cost: 0

Cumulative Re-CPU Cost: 0

Cumulative Re-I/O Cost: 232559

Cumulative First Row Cost: 1.17875e+07

Estimated Bufferpool Buffers: 1.67277e+06

Arguments:

AGGMODE : (Aggregation Mode)
 PARTIAL
 DUPLWARN: (Duplicates Warning flag)
 FALSE
 NUMROWS : (Estimated number of rows)
 30000000
 PARTCOLS: (Table partitioning columns)
 1: Q2.L_ORDERKEY
 ROWWIDTH: (Estimated width of rows)
 25
 SORTKEY : (Sort Key column)
 1: Q2.L_ORDERKEY(A)
 SORTTYPE: (Intra-Partition parallelism sort type)
 PARTITIONED
 SPILLED : (Pages spilled to bufferpool or disk)
 232559
 TEMPSIZE: (Temporary Table Page Size)
 4096
 UNIQUE : (Uniqueness required flag)
 FALSE

Input Streams:

 2) From Operator #12

Estimated number of rows: 1.1997e+08
 Number of columns: 2
 Subquery predicate ID: Not Applicable

Column Names:

 +Q2.L_QUANTITY+Q2.L_ORDERKEY

Output Streams:

 3) To Operator #10

Estimated number of rows: 3e+07
 Number of columns: 2
 Subquery predicate ID: Not Applicable

Column Names:

 +Q2.L_ORDERKEY(A)+Q2.L_QUANTITY

12) IXSCAN: (Index Scan)
 Cumulative Total Cost: 5.75522e+06

Cumulative CPU Cost: 2.48044e+11
Cumulative I/O Cost: 1.44021e+06
Cumulative Re-Total Cost: 5.63989e+06
Cumulative Re-CPU Cost: 1.01213e+10
Cumulative Re-I/O Cost: 1.44021e+06
Cumulative First Row Cost: 100.028
Estimated Bufferpool Buffers: 1.44022e+06

Arguments:

MAXPAGES: (Maximum pages for prefetch)
1411407
PREFETCH: (Type of Prefetch)
SEQUENTIAL
ROWLOCK : (Row Lock intent)
NEXT KEY SHARE
SCANDIR : (Scan Direction)
FORWARD
SCANGRAN: (Intra-Partition Parallelism Scan Granularity)
400
SCANTYPE: (Intra-Partition Parallelism Scan Type)
LOCAL PARALLEL
SCANUNIT: (Intra-Partition Parallelism Scan Unit)
ROW
TABLOCK : (Table Lock intent)
INTENT SHARE

Input Streams:

1) From Object TPCD.L_PKSKOKEPDSQN

Estimated number of rows: 1.1997e+08
Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:

+Q1.\$RID\$+Q1.L_QUANTITY+Q1.L_ORDERKEY

Output Streams:

2) To Operator #11

Estimated number of rows: 1.1997e+08
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.L_QUANTITY+Q2.L_ORDERKEY

13) FETCH : (Fetch)
Cumulative Total Cost: 100.034
Cumulative CPU Cost: 70885.6
Cumulative I/O Cost: 4
Cumulative Re-Total Cost: 50.0211
Cumulative Re-CPU Cost: 43580.6
Cumulative Re-I/O Cost: 2
Cumulative First Row Cost: 100.033
Estimated Bufferpool Buffers: 1.13613e+06

Arguments:

JN INPUT: (Join input leg)
INNER
MAXPAGES: (Maximum pages for prefetch)
1
MAXPAGES: (Maximum pages for prefetch)
1
PREFETCH: (Type of Prefetch)
NONE
ROWLOCK : (Row Lock intent)
NEXT KEY SHARE
TABLOCK : (Table Lock intent)
INTENT SHARE
TBISOLVL: (Table access Isolation Level)
CURSOR STABILITY

Input Streams:

8) From Operator #14

Estimated number of rows: 1
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q4.0_ORDERKEY(A)+Q4.0_ORDERDATE(A)

9) From Object TPCD.ORDERS

Estimated number of rows: 3e+07
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q4.0_TOTALPRICE+Q4.0_CUSTKEY

Output Streams:

10) To Operator #7

Estimated number of rows: 1
Number of columns: 4
Subquery predicate ID: Not Applicable

Column Names:

+Q4.0_ORDERKEY(A)+Q4.0_ORDERDATE(A)
+Q4.0_TOTALPRICE+Q4.0_CUSTKEY

14) IXSCAN: (Index Scan)

Cumulative Total Cost: 75.0263
Cumulative CPU Cost: 54368.6
Cumulative I/O Cost: 3
Cumulative Re-Total Cost: 25.0131
Cumulative Re-CPU Cost: 27063.7
Cumulative Re-I/O Cost: 1
Cumulative First Row Cost: 75.0263
Estimated Bufferpool Buffers: 247938

Arguments:

MAXPAGES: (Maximum pages for prefetch)
1
PREFETCH: (Type of Prefetch)
NONE
ROWLOCK : (Row Lock intent)
NEXT KEY SHARE
SCANDIR : (Scan Direction)
FORWARD
TABLOCK : (Table Lock intent)
INTENT SHARE

Predicates:

7) Start Key Predicate
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 3.33333e-08

Predicate Text:

(Q3.\$C1 = Q4.0_ORDERKEY)

7) Stop Key Predicate
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 3.33333e-08

Predicate Text:

(Q3.\$C1 = Q4.0_ORDERKEY)

Input Streams:

7) From Object TPCD.0_OK_OD_OP

Estimated number of rows: 3e+07
Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:

+Q4.0_ORDERKEY(A)+Q4.0_ORDERDATE(A)+Q4.\$RID\$

Output Streams:

8) To Operator #13

Estimated number of rows: 1
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q4.0_ORDERKEY(A)+Q4.0_ORDERDATE(A)

15) SHIP : (Ship)
Cumulative Total Cost: 68.7753
Cumulative CPU Cost: 49825.4
Cumulative I/O Cost: 2.75005
Cumulative Re-Total Cost: 43.7684
Cumulative Re-CPU Cost: 35520.4
Cumulative Re-I/O Cost: 1.75005
Cumulative First Row Cost: 68.7745
Estimated Bufferpool Buffers: 37191

Remote communication cost:9.35938

Arguments:

CSERQY : (Remote common subexpression)
FALSE

DSTSEVER: (Destination (ship to) server)
- (NULL).

JN INPUT: (Join input leg)
INNER

RMTQTX : (Remote statement)

SELECT AO."C_CUSTKEY", AO."C_NAME" FROM "TPCD"."CUSTOMER" AO WHERE
(AO."C_CUSTKEY" = :H0)

SRCSEVER: (Source (ship from) server)
SQLSERV

STREAM : (Remote stream)
FALSE

Input Streams:

15) From Object MSS.CUSTOMER

Estimated number of rows: 1.5e+06
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q5.C_NAME

Output Streams:

16) To Operator #3

Estimated number of rows: 0.750047
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q5.C_NAME+Q5.C_CUSTKEY

Objects Used in Access Plan:

Schema: TPCD
Name: LINEITEM

Type: Table (reference only)

Schema: TPCD

Name: L_PKSKOKEPDSQN

Type: Index

Time of creation: 2004-06-09-04.46.46.326250

Last statistics update: 2004-06-13-00.38.53.142096

Number of columns: 6

Number of rows: 119969523

Width of rows: -1

Number of buffer pool pages: 4164234

Distinct row values: No

Tablespace name: INDEX_TS

Tablespace overhead: 24.100000

Tablespace transfer rate: 0.900000

Source for statistics: Single Node

Prefetch page count: 96

Container extent page count: 32

Index clustering statistic: 0.000003

Index leaf pages: 1411407

Index tree levels: 5

Index full key cardinality: 119969523

Index first key cardinality: 4000000

Index first 2 keys cardinality: 15871205

Index first 3 keys cardinality: 119969517

Index first 4 keys cardinality: 119969523

Index sequential pages: 1411404

Index page density: 98

Index avg sequential pages: 705702

Index avg gap between sequences:11

Index avg random pages: 0

Fetch avg sequential pages: -1

Fetch avg gap between sequences:-1

Fetch avg random pages: -1

Index RID count: 119969523

Index deleted RID count: 0

Index empty leaf pages: 0

Base Table Schema: TPCD

Base Table Name: LINEITEM

Columns in index:

L_PARTKEY

L_SUPPKEY

L_ORDERKEY

L_EXTENDEDPRICE

L_DISCOUNT

L_QUANTITY

Schema: TPCD

Name: O_OK_OD_OP

Type: Index

- Time of creation: 2004-06-09-04.46.45.691853
- Last statistics update: 2004-06-13-00.52.30.843995
- Number of columns: 3
- Number of rows: 30000000
- Width of rows: -1
- Number of buffer pool pages: 888190
- Distinct row values: Yes
- Tablespace name: INDEX_TS
- Tablespace overhead: 24.100000
- Tablespace transfer rate: 0.900000
- Source for statistics: Single Node
- Prefetch page count: 96
- Container extent page count: 32
- Index clustering statistic: 1.000000
- Index leaf pages: 247935
- Index tree levels: 4
- Index full key cardinality: 30000000
- Index first key cardinality: 30000000
- Index first 2 keys cardinality: 30000000
- Index first 3 keys cardinality: 30000000
- Index first 4 keys cardinality: -1
- Index sequential pages: 247934
- Index page density: 99
- Index avg sequential pages: 247934
- Index avg gap between sequences: 0
- Index avg random pages: 0
- Fetch avg sequential pages: -1
- Fetch avg gap between sequences: -1
- Fetch avg random pages: -1
- Index RID count: 30000000
- Index deleted RID count: 0
- Index empty leaf pages: 0
- Base Table Schema: TPCD
- Base Table Name: ORDERS
- Columns in index:
 - O_ORDERKEY
 - O_ORDERDATE
 - O_ORDERPRIORITY

Schema: MSS

Name: CUSTOMER

Type: Nickname

- Time of creation: 2004-06-10-22.24.53.208724
- Last statistics update:
- Number of columns: 8
- Number of rows: 1500000
- Width of rows: 43
- Number of buffer pool pages: 34412

Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: TPCD

Name: ORDERS

Type: Table

Time of creation: 2004-06-09-04.46.45.050919
Last statistics update: 2004-06-13-00.52.30.843995
Number of columns: 9
Number of rows: 30000000
Width of rows: 32
Number of buffer pool pages: 888190
Distinct row values: No
Tablespace name: DATA_TS
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 96
Container extent page count: 32
Table overflow record count: 0
Table Active Blocks: -1

Base Table For Index Not Already Shown:

Schema: TPCD

Name: LINEITEM

Time of creation: 2004-06-09-04.46.45.763452
Last statistics update: 2004-06-13-00.38.53.142096
Number of columns: 16
Number of rows: 119969523
Number of pages: 4164234
Number of pages with rows: 4164233
Tablespace name: DATA_TS
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Prefetch page count: 96
Container extent page count: 32
Table overflow record count: 0
Indexspace name: INDEX_TS

Access plan description

The access plan graph for our SQL query has been highlighted in Example B-9 on page 490 under “Access Plan”.

Reading this graph bottom up reveals the following:

- ▶ The bottom left part of the graph indicates that the L_PKSKOKEPDSQN index on the LINEITEM table is scanned by the IXSCAN operator 12 and returns 1.1997e+08 rows that includes two columns, L_QUANTITY and L_ORDERKEY (see Output Streams for IXSCAN operator 12), to the SORT operator 11.
- ▶ The 1.1997e+08 rows from the IXSCAN operator 12 operator are input to the SORT operator 11 with two columns (L_QUANTITY and L_ORDERKEY), which are aggregated and sorted in ascending order on L_ORDERKEY (see Arguments in the SORT operator 11) and stored in a temporary table.
- ▶ The 3e+07 sorted rows in the temporary table are then scanned (TBSCAN operator 10) and passed to the GRPBY operator 9.
- ▶ The GRPBY operator 9 aggregates these rows on the L_ORDERKEY column and passes it on to the FILTER operator 8.
- ▶ The FILTER operator 8 applies a residual predicate of L_QUANTITY > 300 and passes the 1272.23 rows as the outer table of a nested loop join (NLJOIN operator 7). The inner table of this merge scan join is estimated to have 1 row (see FETCH operator 13), and the result of the nested loop join is estimated to be 1272.23 rows.
- ▶ The 1 row of the inner table of the nested loop join is formed from a fetch (FETCH operator 13) of an index scan (IXSCAN operator 14) of O_OK_OD_OP index, which returns one row. The predicate is an equality predicate on O_ORDERKEY (see Predicates in IXSCAN operator 14). Four columns are returned to the FETCH operator 13, two from the index (O_ORDERKEY and O_ORDERDATE) and two from the table (O_TOTALPRICE and O_CUSTKEY), as shown in the Input Streams for the FETCH operator 13.
- ▶ The 1272.23 rows are passed to the sort (SORT operator 6), which sorts the data in descending O_TOTALPRICE, ascending O_ORDERDATE, and ascending O_ORDERKEY order, and writes the results to a temporary table.
- ▶ The rows in the temporary table are scanned (TBSCAN operator 5) and passed as the outer table of a nested loop join (NLJOIN operator 3).
- ▶ The listener merge table queue (LMTQ operator 4) indicates that SMP intra-parallelism with a degree of 8 (see TQDEGREE field in Arguments for this operator). Each of these 8 processes performed the set of operations 12, 11, 10, 9, 8, 14, 13, 7, 6, and 5 in parallel, and the final result of 1272.23 rows

was obtained by merging the results from the 8 parallel processes by the LMTQ operator 4.

- ▶ The inner table of the nested loop join (NLJOIN operator 3) is formed by the SHIP operator 15, which retrieves 0.750047 rows from the CUSTOMER table in the SQL server with a cardinality of 1.5e+06. The RMTQTXT field in the SHIP operator 15 shows the equality predicate on C_CUSTKEY being applied at the remote source. Since it is a unique key, a maximum of one row is returned.
- ▶ The 954.234 rows estimated to be output of the nested loop join is passed to the GRPBY operator 2, which groups the rows on five columns (O_ORDERKEY, O_ORDERDATE, O_TOTALPRICE, C_CUSTKEY, and C_NAME). The GRPBY operator 2 estimates that 100 rows will be the result that it passes on to the RETURN operator 1.
- ▶ The total cost is estimated to be 1.30554e+07 timerons, and intra-parallelism is used with a degree of eight (Query Degree:8 and an LMTQ table operator in the Access Plan).
- ▶ Note that the Parallelism field in the Database Context identifies that intra-partition parallelism is enabled for this environment.

Analysis

Important: Bearing in mind that the EXPLAIN output information is DB2 optimizer estimates only and does not necessarily reflect actual runtime metrics, you should be cautious in drawing conclusions about relative costs associated with each operator, as well as relying on the number of estimated rows retrieved or returned by the various operators. EXPLAIN output information is best used in conjunction with runtime metrics gathered through monitoring.

The following observations apply to the **db2exfmt** output shown in Example B-9 on page 490:

- ▶ There is only one SHIP operator to the SQL Server data source and it references the MSS.CUSTOMER nickname. The equality predicate is applied as part of the nested loop join at the remote data source.
- ▶ The cardinality of the nicknames appear to be valid since the DB2 optimizer default of 1000 rows does not appear. This, however, does not guarantee that the DB2 II catalog reflects the actual number of rows at the remote data source.
- ▶ There are a few sorts involved (operators 6, and 11) with large numbers of rows involved. The Database Context shows the Sort Heap size to be 256 pages, while the Buffer Pool size is 1000. This causes spillover to buffer pool

and disk (temporary table space), which can impact performance. SORT operator 6 estimates a need for 237648 buffers, while SORT operator 11 estimates a need for 1.67277e+06 buffers. While this indicates a potential performance improvement to be gained by tuning the sort heap and number of buffers, it appears that a significant portion of the total estimated cost of the query (1.30554e+07) comes from scanning the LINEITEM index, sorting, grouping, and filtering 3e+07 rows for an estimated cost of 1.27556e+07 timerons.

- Note that the LINEITEM table is not directly accessed since all the necessary information is obtained from the index.

Our analysis points to satisfactory exploitation of SMP parallelism, and the need for possible tuning of the sort heap and number of buffers after actual runtime metrics are gathered for this query.

Database Partition Feature (DPF) with FENCED = 'N'

Our objective was to review the access plan in a **db2exfmt** output for a federated query that joined nickname data at a remote data source with local data in a DPF environment with the FENCED = 'N' wrapper option (default, and also known as "trusted" wrapper), and with the FENCED = 'Y' option.

In this scenario, we used the default FENCED = 'N' wrapper option.

We used the 64 bit instance for this example. We set the database manager configuration option INTRA_PARALLEL = NO (intra-partition parallelism disabled).

We used a simple query that joined the customer and order table to obtain a complete list of all orders by customer. In this scenario, the orders table resides locally on a DB2 table, while the customer table resides on Oracle.

Example B-10 shows the complete **db2exfmt** output for our SQL query. The SQL statement we issued has been highlighted under "Original statement" in this output.

Example: B-10 db2exfmt output for trusted wrapper

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

***** EXPLAIN INSTANCE *****

DB2_VERSION: 08.02.0
SOURCE_NAME: SQLC2E05
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-06-23-09.16.49.241800
EXPLAIN_REQUESTER: DB2I64P

Database Context:

Parallelism: Inter-Partition Parallelism
CPU Speed: 5.313873e-07
Comm Speed: 100
Buffer Pool size: 75000
Sort Heap size: 256
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 640

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 203 -----

QUERYNO: 1
QUERYTAG:
Statement Type: Select
Updatable: No
Deletable: No
Query Degree: 1

Original Statement:

select *
from tpcd.orders o, ora.customer c
where o.o_custkey = c.c_custkey

Optimized Statement:

SELECT Q3.\$C7 AS "O_ORDERKEY", Q3.\$C8 AS "O_CUSTKEY", Q3.\$C6 AS
"O_ORDERSTATUS", Q3.\$C5 AS "O_TOTALPRICE", Q3.\$C4 AS "O_ORDERDATE",
Q3.\$C3 AS "O_ORDERPRIORITY", Q3.\$C2 AS "O_CLERK", Q3.\$C1 AS

```

        "O_SHIPPRIORITY", Q3.$C0 AS "O_COMMENT", Q1.C_CUSTKEY AS "C_CUSTKEY",
        Q1.C_NAME AS "C_NAME", Q1.C_ADDRESS AS "C_ADDRESS", Q1.C_NATIONKEY AS
        "C_NATIONKEY", Q1.C_PHONE AS "C_PHONE", Q1.C_ACCTBAL AS "C_ACCTBAL",
        Q1.C_MKTSEGMENT AS "C_MKTSEGMENT", Q1.C_COMMENT AS "C_COMMENT"
FROM ORA.CUSTOMER AS Q1,
    (SELECT Q2.O_COMMENT, Q2.O_SHIPPRIORITY, Q2.O_CLERK, Q2.O_ORDERPRIORITY,
        Q2.O_ORDERDATE, Q2.O_TOTALPRICE, Q2.O_ORDERSTATUS, Q2.O_ORDERKEY,
        Q2.O_CUSTKEY
    FROM TPCD.ORDERS AS Q2) AS Q3
WHERE (Q3.$C8 = Q1.C_CUSTKEY)

```

Access Plan:

```

-----
Total Cost: 3.4124e+07
Query Degree:1

```

```

      Rows
      RETURN
      ( 1)
      Cost
      I/O
      |
      2.27853e+07
      MSJOIN
      ( 2)
      3.4124e+07
      5.62735e+06
      /-----\
      1.5e+06      15.1902
      SHIP      FILTER
      ( 3)      ( 6)
      200146      3.39164e+07
      35041      5.59231e+06
      |
      1.5e+06      3.00114e+07
      NICKNM: ORA      TBSCAN
      CUSTOMER      ( 7)
      3.39164e+07
      5.59231e+06
      |
      3.00114e+07
      SORT
      ( 8)
      2.83444e+07
      2.94425e+06
      |
      3.00114e+07
      DTQ
      ( 9)

```

```

638128
296184
|
1.00038e+07
TBSCAN
( 10)
628410
296184
|
1.00038e+07
TABLE: TPCD
ORDERS

```

1) RETURN: (Return Result)

Cumulative Total Cost: 3.4124e+07
Cumulative CPU Cost: 4.51893e+11
Cumulative I/O Cost: 5.62735e+06
Cumulative Re-Total Cost: 3.4124e+07
Cumulative Re-CPU Cost: 4.51893e+11
Cumulative Re-I/O Cost: 5.62735e+06
Cumulative First Row Cost: 2.8378e+07
Cumulative Comm Cost: 2.41827e+06
Cumulative First Comm Cost: 2.41827e+06
Estimated Bufferpool Buffers: 2.68311e+06
Remote communication cost: 1.18291e+06

Arguments:

```

-----
BLDLEVEL: (Build level)
          DB2 v8.1.1.64 : s040509
STMHEAP: (Statement heap size)
          4096

```

Input Streams:

```

-----
9) From Operator #2

Estimated number of rows: 2.27853e+07
Partition Map ID: -100
Partitioning: (COOR )
              Coordinator Partition
Number of columns: 17
Subquery predicate ID: Not Applicable

```

Column Names:

```

-----

```

```
+Q4.C_COMMENT+Q4.C_MKTSEGMENT+Q4.C_ACCTBAL
+Q4.C_PHONE+Q4.C_NATIONKEY+Q4.C_ADDRESS
+Q4.C_NAME+Q4.C_CUSTKEY+Q4.O_COMMENT
+Q4.O_SHIPPRIORITY+Q4.O_CLERK
+Q4.O_ORDERPRIORITY+Q4.O_ORDERDATE
+Q4.O_TOTALPRICE+Q4.O_ORDERSTATUS+Q4.O_CUSTKEY
+Q4.O_ORDERKEY
```

Partition Column Names:

+NONE

2) MSJOIN: (Merge Scan Join)

```
Cumulative Total Cost: 3.4124e+07
Cumulative CPU Cost: 4.51893e+11
Cumulative I/O Cost: 5.62735e+06
Cumulative Re-Total Cost: 3.4124e+07
Cumulative Re-CPU Cost: 4.51893e+11
Cumulative Re-I/O Cost: 5.62735e+06
Cumulative First Row Cost: 2.8378e+07
Cumulative Comm Cost:2.41827e+06
Cumulative First Comm Cost:2.41827e+06
Estimated Bufferpool Buffers: 2.68311e+06
Remote communication cost:1.18291e+06
```

Arguments:

EARLYOUT: (Early Out flag)

NONE

INNERCOL: (Inner Order Columns)

1: Q3.O_CUSTKEY(A)

OUTERCOL: (Outer Order columns)

1: Q1.C_CUSTKEY(A)

TEMPSIZE: (Temporary Table Page Size)

4096

Predicates:

2) Predicate used in Join

Relational Operator: Equal (=)

Subquery Input Required: No

Filter Factor: 5.06148e-07

Predicate Text:

(Q3.\$C8 = Q1.C_CUSTKEY)

Input Streams:

2) From Operator #3

Estimated number of rows: 1.5e+06
Partition Map ID: -100
Partitioning: (COOR)
Coordinator Partition
Number of columns: 8
Subquery predicate ID: Not Applicable

Column Names:

+Q1.C_CUSTKEY(A)+Q1.C_COMMENT+Q1.C_MKTSEGMENT
+Q1.C_ACCTBAL+Q1.C_PHONE+Q1.C_NATIONKEY
+Q1.C_ADDRESS+Q1.C_NAME

Partition Column Names:

+NONE

8) From Operator #6

Estimated number of rows: 15.1902
Partition Map ID: -100
Partitioning: (COOR)
Coordinator Partition
Number of columns: 9
Subquery predicate ID: Not Applicable

Column Names:

+Q3.0_CUSTKEY(A)+Q3.0_COMMENT
+Q3.0_SHIPPRIORITY+Q3.0_CLERK
+Q3.0_ORDERPRIORITY+Q3.0_ORDERDATE
+Q3.0_TOTALPRICE+Q3.0_ORDERSTATUS+Q3.0_ORDERKEY

Partition Column Names:

+NONE

Output Streams:

9) To Operator #1

Estimated number of rows: 2.27853e+07
Partition Map ID: -100
Partitioning: (COOR)

Coordinator Partition
Number of columns: 17
Subquery predicate ID: Not Applicable

Column Names:

+Q4.C_COMMENT+Q4.C_MKTSEGMENT+Q4.C_ACCTBAL
+Q4.C_PHONE+Q4.C_NATIONKEY+Q4.C_ADDRESS
+Q4.C_NAME+Q4.C_CUSTKEY+Q4.O_COMMENT
+Q4.O_SHIPPRIORITY+Q4.O_CLERK
+Q4.O_ORDERPRIORITY+Q4.O_ORDERDATE
+Q4.O_TOTALPRICE+Q4.O_ORDERSTATUS+Q4.O_CUSTKEY
+Q4.O_ORDERKEY

Partition Column Names:

+NONE

3) SHIP : (Ship)

Cumulative Total Cost: 200146
Cumulative CPU Cost: 4.18261e+09
Cumulative I/O Cost: 35041
Cumulative Re-Total Cost: 2097.12
Cumulative Re-CPU Cost: 3.94651e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 75.0424
Cumulative Comm Cost:0
Cumulative First Comm Cost:0
Estimated Bufferpool Buffers: 35042
Remote communication cost:1.18291e+06

Arguments:

CSERQY : (Remote common subexpression)

FALSE

DSTSEVER: (Destination (ship to) server)

- (NULL).

JN INPUT: (Join input leg)

OUTER

RMTQTX : (Remote statement)

SELECT AO."C_CUSTKEY", AO."C_NAME", AO."C_ADDRESS", AO."C_NATIONKEY",
AO."C_PHONE", AO."C_ACCTBAL", AO."C_MKTSEGMENT", AO."C_COMMENT" FROM
"IITEST"."CUSTOMER" AO ORDER BY 1 ASC

SRCSEVER: (Source (ship from) server)

ORASERV

STREAM : (Remote stream)

FALSE

Input Streams:

1) From Object ORA.CUSTOMER

Estimated number of rows: 1.5e+06
Partition Map ID: -100
Partitioning: (COOR)
Coordinator Partition
Number of columns: 7
Subquery predicate ID: Not Applicable

Column Names:

+Q1.C_COMMENT+Q1.C_MKTSEGMENT+Q1.C_ACCTBAL
+Q1.C_PHONE+Q1.C_NATIONKEY+Q1.C_ADDRESS
+Q1.C_NAME

Partition Column Names:

+NONE

Output Streams:

2) To Operator #2

Estimated number of rows: 1.5e+06
Partition Map ID: -100
Partitioning: (COOR)
Coordinator Partition
Number of columns: 8
Subquery predicate ID: Not Applicable

Column Names:

+Q1.C_CUSTKEY(A)+Q1.C_COMMENT+Q1.C_MKTSEGMENT
+Q1.C_ACCTBAL+Q1.C_PHONE+Q1.C_NATIONKEY
+Q1.C_ADDRESS+Q1.C_NAME

Partition Column Names:

+NONE

6) FILTER: (Filter)

Cumulative Total Cost: 3.39164e+07
Cumulative CPU Cost: 4.33688e+11
Cumulative I/O Cost: 5.59231e+06
Cumulative Re-Total Cost: 5.57199e+06

Cumulative Re-CPU Cost: 8.3105e+10
Cumulative Re-I/O Cost: 2.64806e+06
Cumulative First Row Cost: 2.83779e+07
Cumulative Comm Cost: 2.41827e+06
Cumulative First Comm Cost: 2.41827e+06
Estimated Bufferpool Buffers: 2.64806e+06

Arguments:

JN INPUT: (Join input leg)
INNER

Predicates:

2) Residual Predicate
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 5.06148e-07

Predicate Text:

(Q3.\$C8 = Q1.C_CUSTKEY)

Input Streams:

7) From Operator #7

Estimated number of rows: 3.00114e+07
Partition Map ID: -100
Partitioning: (COOR)
Coordinator Partition
Number of columns: 9
Subquery predicate ID: Not Applicable

Column Names:

+Q3.0_CUSTKEY(A)+Q3.0_COMMENT
+Q3.0_SHIPPRIORITY+Q3.0_CLERK
+Q3.0_ORDERPRIORITY+Q3.0_ORDERDATE
+Q3.0_TOTALPRICE+Q3.0_ORDERSTATUS+Q3.0_ORDERKEY

Partition Column Names:

+NONE

Output Streams:

8) To Operator #2

Estimated number of rows: 15.1902
Partition Map ID: -100
Partitioning: (COOR)
Coordinator Partition
Number of columns: 9
Subquery predicate ID: Not Applicable

Column Names:

+Q3.0_CUSTKEY(A)+Q3.0_COMMENT
+Q3.0_SHIPRIORITY+Q3.0_CLERK
+Q3.0_ORDERPRIORITY+Q3.0_ORDERDATE
+Q3.0_TOTALPRICE+Q3.0_ORDERSTATUS+Q3.0_ORDERKEY

Partition Column Names:

+NONE

7) TBSCAN: (Table Scan)

Cumulative Total Cost: 3.39164e+07
Cumulative CPU Cost: 4.33688e+11
Cumulative I/O Cost: 5.59231e+06
Cumulative Re-Total Cost: 5.57199e+06
Cumulative Re-CPU Cost: 8.3105e+10
Cumulative Re-I/O Cost: 2.64806e+06
Cumulative First Row Cost: 2.83779e+07
Cumulative Comm Cost: 2.41827e+06
Cumulative First Comm Cost: 2.41827e+06
Estimated Bufferpool Buffers: 2.64806e+06

Arguments:

MAXPAGES: (Maximum pages for prefetch)

ALL

PREFETCH: (Type of Prefetch)

SEQUENTIAL

SCANDIR : (Scan Direction)

FORWARD

Input Streams:

6) From Operator #8

Estimated number of rows: 3.00114e+07
Partition Map ID: -100
Partitioning: (COOR)

Coordinator Partition
Number of columns: 9
Subquery predicate ID: Not Applicable

Column Names:

+Q3.0_CUSTKEY(A)+Q3.0_COMMENT
+Q3.0_SHIPRIORITY+Q3.0_CLERK
+Q3.0_ORDERPRIORITY+Q3.0_ORDERDATE
+Q3.0_TOTALPRICE+Q3.0_ORDERSTATUS+Q3.0_ORDERKEY

Partition Column Names:

+NONE

Output Streams:

7) To Operator #6

Estimated number of rows: 3.00114e+07
Partition Map ID: -100
Partitioning: (COOR)
Coordinator Partition
Number of columns: 9
Subquery predicate ID: Not Applicable

Column Names:

+Q3.0_CUSTKEY(A)+Q3.0_COMMENT
+Q3.0_SHIPRIORITY+Q3.0_CLERK
+Q3.0_ORDERPRIORITY+Q3.0_ORDERDATE
+Q3.0_TOTALPRICE+Q3.0_ORDERSTATUS+Q3.0_ORDERKEY

Partition Column Names:

+NONE

8) SORT : (Sort)

Cumulative Total Cost: 2.83444e+07
Cumulative CPU Cost: 3.50582e+11
Cumulative I/O Cost: 2.94425e+06
Cumulative Re-Total Cost: 0
Cumulative Re-CPU Cost: 0
Cumulative Re-I/O Cost: 2.64806e+06
Cumulative First Row Cost: 2.83444e+07
Cumulative Comm Cost: 2.41827e+06
Cumulative First Comm Cost: 2.41827e+06

Estimated Bufferpool Buffers: 2.94425e+06

Arguments:

DUPLWARN: (Duplicates Warning flag)

FALSE

NUMROWS : (Estimated number of rows)

30011368

ROWWIDTH: (Estimated width of rows)

112

SORTKEY : (Sort Key column)

1: Q3.0_CUSTKEY(A)

SPILLED : (Pages spilled to bufferpool or disk)

2.64806e+06

TEMPSIZE: (Temporary Table Page Size)

4096

UNIQUE : (Uniqueness required flag)

FALSE

Input Streams:

5) From Operator #9

Estimated number of rows: 3.00114e+07

Partition Map ID: -100

Partitioning: (COOR)

Coordinator Partition

Number of columns: 9

Subquery predicate ID: Not Applicable

Column Names:

+Q3.0_COMMENT+Q3.0_SHIPPRIORITY+Q3.0_CLERK

+Q3.0_ORDERPRIORITY+Q3.0_ORDERDATE

+Q3.0_TOTALPRICE+Q3.0_ORDERSTATUS

+Q3.0_ORDERKEY+Q3.0_CUSTKEY

Partition Column Names:

+NONE

Output Streams:

6) To Operator #7

Estimated number of rows: 3.00114e+07

Partition Map ID: -100

Partitioning: (COOR)

Coordinator Partition
Number of columns: 9
Subquery predicate ID: Not Applicable

Column Names:

+Q3.0_CUSTKEY(A)+Q3.0_COMMENT
+Q3.0_SHIPPRIORITY+Q3.0_CLERK
+Q3.0_ORDERPRIORITY+Q3.0_ORDERDATE
+Q3.0_TOTALPRICE+Q3.0_ORDERSTATUS+Q3.0_ORDERKEY

Partition Column Names:

+NONE

9) TQ : (Table Queue)

Cumulative Total Cost: 638128
Cumulative CPU Cost: 3.73425e+10
Cumulative I/O Cost: 296184
Cumulative Re-Total Cost: 628410
Cumulative Re-CPU Cost: 1.90555e+10
Cumulative Re-I/O Cost: 296184
Cumulative First Row Cost: 10.4828
Cumulative Comm Cost: 2.41827e+06
Cumulative First Comm Cost: 0
Estimated Bufferpool Buffers: 296184

Arguments:

LISTENER: (Listener Table Queue type)
FALSE
TQMERGE : (Merging Table Queue flag)
FALSE
TQREAD : (Table Queue Read type)
READ AHEAD
TQSEND : (Table Queue Write type)
DIRECTED
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

4) From Operator #10

Estimated number of rows: 1.00038e+07
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions

Number of columns: 9
Subquery predicate ID: Not Applicable

Column Names:

+Q3.0_COMMENT+Q3.0_SHIPPRIORITY+Q3.0_CLERK
+Q3.0_ORDERPRIORITY+Q3.0_ORDERDATE
+Q3.0_TOTALPRICE+Q3.0_ORDERSTATUS
+Q3.0_ORDERKEY+Q3.0_CUSTKEY

Partition Column Names:

+1: Q3.0_ORDERKEY

Output Streams:

5) To Operator #8

Estimated number of rows: 3.00114e+07
Partition Map ID: -100
Partitioning: (COOR)
Coordinator Partition
Number of columns: 9
Subquery predicate ID: Not Applicable

Column Names:

+Q3.0_COMMENT+Q3.0_SHIPPRIORITY+Q3.0_CLERK
+Q3.0_ORDERPRIORITY+Q3.0_ORDERDATE
+Q3.0_TOTALPRICE+Q3.0_ORDERSTATUS
+Q3.0_ORDERKEY+Q3.0_CUSTKEY

Partition Column Names:

+NONE

10) TBSCAN: (Table Scan)

Cumulative Total Cost: 628410
Cumulative CPU Cost: 1.90555e+10
Cumulative I/O Cost: 296184
Cumulative Re-Total Cost: 628410
Cumulative Re-CPU Cost: 1.90555e+10
Cumulative Re-I/O Cost: 296184
Cumulative First Row Cost: 10.4278
Cumulative Comm Cost:0
Cumulative First Comm Cost:0
Estimated Bufferpool Buffers: 296184

Arguments:

MAXPAGES: (Maximum pages for prefetch)
ALL
PREFETCH: (Type of Prefetch)
SEQUENTIAL
ROWLOCK : (Row Lock intent)
NEXT KEY SHARE
SCANDIR : (Scan Direction)
FORWARD
TABLOCK : (Table Lock intent)
INTENT SHARE
TBISOLVL: (Table access Isolation Level)
CURSOR STABILITY

Input Streams:

3) From Object TPCD.ORDERS

Estimated number of rows: 1.00038e+07
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 10
Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$+Q2.0_CUSTKEY+Q2.0_ORDERKEY
+Q2.0_ORDERSTATUS+Q2.0_TOTALPRICE
+Q2.0_ORDERDATE+Q2.0_ORDERPRIORITY+Q2.0_CLERK
+Q2.0_SHIPPRIORITY+Q2.0_COMMENT

Partition Column Names:

+1: Q2.0_ORDERKEY

Output Streams:

4) To Operator #9

Estimated number of rows: 1.00038e+07
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 9
Subquery predicate ID: Not Applicable

Column Names:

+Q3.0_COMMENT+Q3.0_SHIPPRIORITY+Q3.0_CLERK
+Q3.0_ORDERPRIORITY+Q3.0_ORDERDATE
+Q3.0_TOTALPRICE+Q3.0_ORDERSTATUS
+Q3.0_ORDERKEY+Q3.0_CUSTKEY

Partition Column Names:

+1: Q3.0_ORDERKEY

Objects Used in Access Plan:

Schema: ORA

Name: CUSTOMER

Type: Nickname

Time of creation: 2004-06-10-08.32.16.950465

Last statistics update:

Number of columns: 8

Number of rows: 1500000

Width of rows: 236

Number of buffer pool pages: 32173

Distinct row values: No

Tablespace name:

Tablespace overhead: 24.100000

Tablespace transfer rate: 0.900000

Source for statistics: Single Node

Prefetch page count: 32

Container extent page count: 32

Schema: TPCD

Name: ORDERS

Type: Table

Time of creation: 2004-06-04-07.02.15.736633

Last statistics update: 2004-06-09-23.20.10.138687

Number of columns: 9

Number of rows: 10003789

Width of rows: 115

Number of buffer pool pages: 296184

Distinct row values: No

Tablespace name: DATA_TS

Tablespace overhead: 9.500000

Tablespace transfer rate: 0.900000

Source for statistics: Single Node

Prefetch page count: 48

Container extent page count: 16

Access plan description

The access plan graph for our SQL query has been highlighted in Example B-10 on page 516 under “Access Plan”.

Reading this graph bottom up reveals the following:

- ▶ The bottom left part of the graph indicates that the ORA.CUSTOMER nickname has 1.5 million rows according to the DB2 II catalog statistics, and that all these rows are processed by the SHIP operator 3. Seven columns (see the Input Streams for the SHIP operator 3) were retrieved.
- ▶ The 1.5e+06 rows from the SHIP operator 3 form the outer table (see the Arguments section of this operator) of a merge scan join (MSJOIN operator 2). The inner table of this merge scan join is estimated to have 15.1902 rows, and the result of the merge scan join is estimated to be 2.27853e+07 rows.
- ▶ The 15.1902 rows for the inner table of the join are formed from a series of operator executions involving a TBSCAN (operator 10), DTQ (operator 9), SORT (operator 8), TBSCAN (operator 7), SORT (operator 8), and FILTER (operator 6).
 - The directed table queue operator (DTQ operator 9) indicates that the rows of the TPCD.ORDERS table are scanned in parallel in each partition (cardinality of 1.00038e+07 rows), and 3.00114e+07 rows (which are the sum of all rows from the three partitions where this table resides) are directed to the coordinator partition (see Output Streams of the DTQ operator 9) for sorting (SORT operator 8) and storing in a temporary table.
 - The TBSCAN operator 7 then scans the temporary table and passes it to the FILTER operator 6, which applies the residual predicate matching the customer key as part of the merge scan join (MSJOIN operator 2).
- ▶ The estimated 2.27853e+07 rows from the merge scan result are passed to the RETURN operator 1.
- ▶ The total cost is estimated to be 3.4124e+07 timerons, and intra-parallelism is not used (Query Degree:1). However, inter-partition parallelism is used since the DTQ operator appears in the access plan.
- ▶ Note that the Parallelism field in the Database Context identifies that inter-partition parallelism is enabled for this environment.

Analysis

Important: Bearing in mind that the EXPLAIN output information is DB2 optimizer estimates only and does not necessarily reflect actual runtime metrics, you should be cautious in drawing conclusions about relative costs associated with each operator, as well as relying on the number of estimated rows retrieved or returned by the various operators. EXPLAIN output information is best used in conjunction with runtime metrics gathered through monitoring.

The following observations apply to the **db2exfmt** output shown in Example B-10 on page 516.

- ▶ There is only one SHIP operator (3) to the Oracle data source, and it references the ORA.CUSTOMER nickname. The RMTQTXT field in the Arguments section of the SHIP operator 3 shows the ORDER BY being pushed down, which indicates a matching collating sequence between the federated server and the Oracle data source.
- ▶ The cardinality of the nicknames appear to be valid since the DB2 optimizer default of 1000 rows does not appear. This, however, does not guarantee that the DB2 II catalog reflects the actual number of rows at the remote data source.
- ▶ There is one sort involved (operators 8) involving 30 million rows. The Database Context shows the Sort Heap size to be 256 pages, while the Buffer Pool size is 75000. This causes spillover to buffer pool and disk (temporary table space), which can impact performance. SORT operator 8 estimates a need for 2,944250 buffers. Since a significant portion of the estimated cost of the query (3.4124e+07) appears to be related to the SORT operator 8, it indicates that potential performance improvements can be gained by tuning the sort heap and number of buffers.
- ▶ A key point is that DB2 II serially performs the join between the local orders table and the customer nickname at the coordinator partition. This is because of the use of the trusted wrapper, which inhibits DB2 II from distributing nickname data across multiple partitions to create a parallel join.

Note: There is no indication of the wrapper type (trusted or not) in the **db2exfmt** output.

The wrapper type is stored in DB2 system catalog table SYSIBM.SYSWRAPOPTIONS or view SYSCAT.WRAPOPTIONS.

Our analysis points to considering the use of other than a trusted wrapper (see “Database Partition Feature (DPF) with FENCED = ‘Y’” on page 534) for greater

parallelism, if appropriate, and the need for possible tuning of the sort heap and number of buffers after actual runtime metrics are gathered for this query.

Database Partition Feature (DPF) with FENCED = 'Y'

In this scenario, we ran the same query and environment used in “Database Partition Feature (DPF) with FENCED = 'N'” on page 516, with the only difference being that the wrapper option was set to FENCED = 'Y'.

Example B-11 shows the complete **db2exfmt** output for our SQL query. The SQL statement we issued has been highlighted under “Original statement” in this output.

Example: B-11 db2exfmt output for fenced wrapper

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

***** EXPLAIN INSTANCE *****

DB2_VERSION: 08.02.0
SOURCE_NAME: SQLC2E05
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-06-23-09.34.25.261504
EXPLAIN_REQUESTER: DB2I64P

Database Context:

Parallelism: Inter-Partition Parallelism
CPU Speed: 5.313873e-07
Comm Speed: 100
Buffer Pool size: 75000
Sort Heap size: 256
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 640

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors

Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 203 -----

QUERYNO: 1
QUERYTAG:
Statement Type: Select
Updatable: No
Deletable: No
Query Degree: 1

Original Statement:

select *
from tpcd.orders o, ora.customer c
where o.o_custkey = c.c_custkey

Optimized Statement:

SELECT Q2.O_ORDERKEY AS "O_ORDERKEY", Q2.O_CUSTKEY AS "O_CUSTKEY",
Q2.O_ORDERSTATUS AS "O_ORDERSTATUS", Q2.O_TOTALPRICE AS
"O_TOTALPRICE", Q2.O_ORDERDATE AS "O_ORDERDATE", Q2.O_ORDERPRIORITY
AS "O_ORDERPRIORITY", Q2.O_CLERK AS "O_CLERK", Q2.O_SHIPPRIORITY AS
"O_SHIPPRIORITY", Q2.O_COMMENT AS "O_COMMENT", Q1.C_CUSTKEY AS
"C_CUSTKEY", Q1.C_NAME AS "C_NAME", Q1.C_ADDRESS AS "C_ADDRESS",
Q1.C_NATIONKEY AS "C_NATIONKEY", Q1.C_PHONE AS "C_PHONE",
Q1.C_ACCTBAL AS "C_ACCTBAL", Q1.C_MKTSEGMENT AS "C_MKTSEGMENT",
Q1.C_COMMENT AS "C_COMMENT"
FROM ORA.CUSTOMER AS Q1, TPCD.ORDERS AS Q2
WHERE (Q2.O_CUSTKEY = Q1.C_CUSTKEY)

Access Plan:

Total Cost: 214094
Query Degree:1

Rows
RETURN
(1)
Cost
I/O
|
2.27853e+07
DTQ
(2)
214094
32193.1

```

      |
      7.5951e+06
      NLJOIN
      ( 3)
      203003
      32193.1
    /-----+-----\
    1.5e+06          5.0634
    BTQ              FETCH
    ( 4)             ( 7)
    128671           53.8812
    32173            17.5276
    |               /-----\
    1.5e+06          5.0634      1.00038e+07
    SHIP            RIDSCN      TABLE: TPCD
    ( 5)            ( 8)        ORDERS
    127364          20.8488
    32173            2
    |               |
    1.5e+06          5.0634
    NICKNM: ORA      SORT
    CUSTOMER         ( 9)
                   20.8481
                   2
                   |
                   5.0634
                   IXSCAN
                   ( 10)
                   20.8465
                   2
                   |
                   1.00038e+07
                   INDEX: TPCD
                   0_CK

```

1) RETURN: (Return Result)

Cumulative Total Cost: 214094

Cumulative CPU Cost: 1.6588e+11

Cumulative I/O Cost: 32193.1

Cumulative Re-Total Cost: 75642.7

Cumulative Re-CPU Cost: 1.42349e+11

Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 56.3935

Cumulative Comm Cost: 6.18905e+06

Cumulative First Comm Cost: 0

Estimated Bufferpool Buffers: 32191.1

Remote communication cost:1.18291e+06

Arguments:

BLDLEVEL: (Build level)

DB2 v8.1.1.64 : s040509

STMHEAP: (Statement heap size)

4096

Input Streams:

11) From Operator #2

Estimated number of rows: 2.27853e+07

Partition Map ID: -100

Partitioning: (COOR)

Coordinator Partition

Number of columns: 17

Subquery predicate ID: Not Applicable

Column Names:

+Q3.C_COMMENT+Q3.C_MKTSEGMENT+Q3.C_ACCTBAL

+Q3.C_PHONE+Q3.C_NATIONKEY+Q3.C_ADDRESS

+Q3.C_NAME+Q3.C_CUSTKEY+Q3.O_COMMENT

+Q3.O_SHIPPRIORITY+Q3.O_CLERK

+Q3.O_ORDERPRIORITY+Q3.O_ORDERDATE

+Q3.O_TOTALPRICE+Q3.O_ORDERSTATUS+Q3.O_CUSTKEY

+Q3.O_ORDERKEY

Partition Column Names:

+NONE

2) TQ : (Table Queue)

Cumulative Total Cost: 214094

Cumulative CPU Cost: 1.6588e+11

Cumulative I/O Cost: 32193.1

Cumulative Re-Total Cost: 75642.7

Cumulative Re-CPU Cost: 1.42349e+11

Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 56.3935

Cumulative Comm Cost:6.18905e+06

Cumulative First Comm Cost:0

Estimated Bufferpool Buffers: 32191.1

Remote communication cost:1.18291e+06

Arguments:

```

-----
LISTENER: (Listener Table Queue type)
FALSE
TQMERGE : (Merging Table Queue flag)
FALSE
TQREAD  : (Table Queue Read type)
READ AHEAD
TQSEND  : (Table Queue Write type)
DIRECTED
UNIQUE  : (Uniqueness required flag)
FALSE

Input Streams:
-----
10) From Operator #3

Estimated number of rows: 7.5951e+06
Partition Map ID: 4
Partitioning: (MULT )
Multiple Partitions
Number of columns: 17
Subquery predicate ID: Not Applicable

Column Names:
-----
+Q3.C_COMMENT+Q3.C_MKTSEGMENT+Q3.C_ACCTBAL
+Q3.C_PHONE+Q3.C_NATIONKEY+Q3.C_ADDRESS
+Q3.C_NAME+Q3.C_CUSTKEY+Q3.O_COMMENT
+Q3.O_SHIPPRIORITY+Q3.O_CLERK
+Q3.O_ORDERPRIORITY+Q3.O_ORDERDATE
+Q3.O_TOTALPRICE+Q3.O_ORDERSTATUS+Q3.O_CUSTKEY
+Q3.O_ORDERKEY

Partition Column Names:
-----
+1: Q3.O_ORDERKEY

Output Streams:
-----
11) To Operator #1

Estimated number of rows: 2.27853e+07
Partition Map ID: -100
Partitioning: (COOR )
Coordinator Partition
Number of columns: 17
Subquery predicate ID: Not Applicable

```


Column Names:

+Q3.C_COMMENT+Q3.C_MKTSEGMENT+Q3.C_ACCTBAL
+Q3.C_PHONE+Q3.C_NATIONKEY+Q3.C_ADDRESS
+Q3.C_NAME+Q3.C_CUSTKEY+Q3.O_COMMENT
+Q3.O_SHIPPRIORITY+Q3.O_CLERK
+Q3.O_ORDERPRIORITY+Q3.O_ORDERDATE
+Q3.O_TOTALPRICE+Q3.O_ORDERSTATUS+Q3.O_CUSTKEY
+Q3.O_ORDERKEY

Partition Column Names:

+NONE

3) NLJOIN: (Nested Loop Join)

Cumulative Total Cost: 203003
Cumulative CPU Cost: 1.45008e+11
Cumulative I/O Cost: 32193.1
Cumulative Re-Total Cost: 75642.7
Cumulative Re-CPU Cost: 1.42349e+11
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 56.3385
Cumulative Comm Cost: 743805
Cumulative First Comm Cost: 0
Estimated Bufferpool Buffers: 32191.1
Remote communication cost: 1.18291e+06

Arguments:

EARLYOUT: (Early Out flag)

NONE

FETCHMAX: (Override for FETCH MAXPAGES)

IGNORE

ISCANMAX: (Override for ISCAN MAXPAGES)

IGNORE

Predicates:

2) Predicate used in Join

Relational Operator: Equal (=)

Subquery Input Required: No

Filter Factor: 5.06148e-07

Predicate Text:

(Q2.O_CUSTKEY = Q1.C_CUSTKEY)

Input Streams:

3) From Operator #4

Estimated number of rows: 1.5e+06
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 8
Subquery predicate ID: Not Applicable

Column Names:

+Q1.C_COMMENT+Q1.C_MKTSEGMENT+Q1.C_ACCTBAL
+Q1.C_PHONE+Q1.C_NATIONKEY+Q1.C_ADDRESS
+Q1.C_NAME+Q1.C_CUSTKEY

Partition Column Names:

+NONE

9) From Operator #7

Estimated number of rows: 5.0634
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 10
Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$+Q2.0_COMMENT+Q2.0_SHIPPRIORITY
+Q2.0_CLERK+Q2.0_ORDERPRIORITY+Q2.0_ORDERDATE
+Q2.0_TOTALPRICE+Q2.0_ORDERSTATUS
+Q2.0_ORDERKEY+Q2.0_CUSTKEY

Partition Column Names:

+1: Q2.0_ORDERKEY

Output Streams:

10) To Operator #2

Estimated number of rows: 7.5951e+06
Partition Map ID: 4
Partitioning: (MULT)

Multiple Partitions
Number of columns: 17
Subquery predicate ID: Not Applicable

Column Names:

+Q3.C_COMMENT+Q3.C_MKTSEGMENT+Q3.C_ACCTBAL
+Q3.C_PHONE+Q3.C_NATIONKEY+Q3.C_ADDRESS
+Q3.C_NAME+Q3.C_CUSTKEY+Q3.O_COMMENT
+Q3.O_SHIPPRIORITY+Q3.O_CLERK
+Q3.O_ORDERPRIORITY+Q3.O_ORDERDATE
+Q3.O_TOTALPRICE+Q3.O_ORDERSTATUS+Q3.O_CUSTKEY
+Q3.O_ORDERKEY

Partition Column Names:

+1: Q3.O_ORDERKEY

4) TQ : (Table Queue)

Cumulative Total Cost: 128671
Cumulative CPU Cost: 5.24183e+09
Cumulative I/O Cost: 32173
Cumulative Re-Total Cost: 1372.57
Cumulative Re-CPU Cost: 2.583e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 25.0828
Cumulative Comm Cost: 743805
Cumulative First Comm Cost: 0
Estimated Bufferpool Buffers: 32173
Remote communication cost: 1.18291e+06

Arguments:

JN INPUT: (Join input leg)
OUTER
LISTENER: (Listener Table Queue type)
FALSE
TQMERGE : (Merging Table Queue flag)
FALSE
TQREAD : (Table Queue Read type)
READ AHEAD
TQSEND : (Table Queue Write type)
BROADCAST
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

2) From Operator #5

Estimated number of rows: 1.5e+06
Partition Map ID: -100
Partitioning: (COOR)
Coordinator Partition
Number of columns: 8
Subquery predicate ID: Not Applicable

Column Names:

+Q1.C_COMMENT+Q1.C_MKTSEGMENT+Q1.C_ACCTBAL
+Q1.C_PHONE+Q1.C_NATIONKEY+Q1.C_ADDRESS
+Q1.C_NAME+Q1.C_CUSTKEY

Partition Column Names:

+NONE

Output Streams:

3) To Operator #3

Estimated number of rows: 1.5e+06
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 8
Subquery predicate ID: Not Applicable

Column Names:

+Q1.C_COMMENT+Q1.C_MKTSEGMENT+Q1.C_ACCTBAL
+Q1.C_PHONE+Q1.C_NATIONKEY+Q1.C_ADDRESS
+Q1.C_NAME+Q1.C_CUSTKEY

Partition Column Names:

+NONE

5) SHIP : (Ship)

Cumulative Total Cost: 127364
Cumulative CPU Cost: 2.78171e+09
Cumulative I/O Cost: 32173
Cumulative Re-Total Cost: 1372.57
Cumulative Re-CPU Cost: 2.583e+09
Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 25.0278
Cumulative Comm Cost:0
Cumulative First Comm Cost:0
Estimated Bufferpool Buffers: 32173
Remote communication cost:1.18291e+06

Arguments:

CSERQY : (Remote common subexpression)
FALSE
DSTSEVER: (Destination (ship to) server)
- (NULL).
RMTQTX : (Remote statement)
SELECT AO."C_CUSTKEY", AO."C_NAME", AO."C_ADDRESS", AO."C_NATIONKEY",
AO."C_PHONE", AO."C_ACCTBAL", AO."C_MKTSEGMENT", AO."C_COMMENT" FROM
"IITEST"."CUSTOMER" AO
SRCSEVER: (Source (ship from) server)
ORASERV
STREAM : (Remote stream)
FALSE

Input Streams:

1) From Object ORA.CUSTOMER

Estimated number of rows: 1.5e+06
Partition Map ID: -100
Partitioning: (COOR)
Coordinator Partition
Number of columns: 9
Subquery predicate ID: Not Applicable

Column Names:

+Q1.\$RID\$+Q1.C_COMMENT+Q1.C_MKTSEGMENT
+Q1.C_ACCTBAL+Q1.C_PHONE+Q1.C_NATIONKEY
+Q1.C_ADDRESS+Q1.C_NAME+Q1.C_CUSTKEY

Partition Column Names:

+NONE

Output Streams:

2) To Operator #4

Estimated number of rows: 1.5e+06
Partition Map ID: -100

Partitioning: (COOR)
Coordinator Partition
Number of columns: 8
Subquery predicate ID: Not Applicable

Column Names:

+Q1.C_COMMENT+Q1.C_MKTSEGMENT+Q1.C_ACCTBAL
+Q1.C_PHONE+Q1.C_NATIONKEY+Q1.C_ADDRESS
+Q1.C_NAME+Q1.C_CUSTKEY

Partition Column Names:

+NONE

7) FETCH : (Fetch)

Cumulative Total Cost: 53.8812
Cumulative CPU Cost: 200124
Cumulative I/O Cost: 17.5276
Cumulative Re-Total Cost: 43.4524
Cumulative Re-CPU Cost: 145972
Cumulative Re-I/O Cost: 15.5276
Cumulative First Row Cost: 31.2557
Cumulative Comm Cost:0
Cumulative First Comm Cost:0
Estimated Bufferpool Buffers: 18.0566

Arguments:

JN INPUT: (Join input leg)
INNER
MAX RIDS: (Maximum RIDs per list prefetch request)
512
MAXPAGES: (Maximum pages for prefetch)
3
PREFETCH: (Type of Prefetch)
LIST
ROWLOCK : (Row Lock intent)
NEXT KEY SHARE
TABLOCK : (Table Lock intent)
INTENT SHARE
TBISOLVL: (Table access Isolation Level)
CURSOR STABILITY

Predicates:

2) Sargable Predicate
Relational Operator: Equal (=)

Subquery Input Required: No
Filter Factor: 5.06148e-07

Predicate Text:

(Q2.O_CUSTKEY = Q1.C_CUSTKEY)

Input Streams:

7) From Operator #8

Estimated number of rows: 5.0634
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$(A)

Partition Column Names:

+1: Q2.O_ORDERKEY

8) From Object TPCD.ORDERS

Estimated number of rows: 1.00038e+07
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 9
Subquery predicate ID: Not Applicable

Column Names:

+Q2.O_COMMENT+Q2.O_SHIPRIORITY+Q2.O_CLERK
+Q2.O_ORDERPRIORITY+Q2.O_ORDERDATE
+Q2.O_TOTALPRICE+Q2.O_ORDERSTATUS
+Q2.O_ORDERKEY+Q2.O_CUSTKEY

Partition Column Names:

+1: Q2.O_ORDERKEY

Output Streams:

9) To Operator #3

Estimated number of rows: 5.0634
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 10
Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$+Q2.0_COMMENT+Q2.0_SHIPPRIORITY
+Q2.0_CLERK+Q2.0_ORDERPRIORITY+Q2.0_ORDERDATE
+Q2.0_TOTALPRICE+Q2.0_ORDERSTATUS
+Q2.0_ORDERKEY+Q2.0_CUSTKEY

Partition Column Names:

+1: Q2.0_ORDERKEY

8) RIDSCN: (Row Identifier Scan)

Cumulative Total Cost: 20.8488
Cumulative CPU Cost: 91759.9
Cumulative I/O Cost: 2
Cumulative Re-Total Cost: 10.4222
Cumulative Re-CPU Cost: 41857
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 20.8481
Cumulative Comm Cost: 0
Cumulative First Comm Cost: 0
Estimated Bufferpool Buffers: 3

Arguments:

NUMROWS : (Estimated number of rows)
6

Input Streams:

6) From Operator #9

Estimated number of rows: 5.0634
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$(A)

Partition Column Names:

+1: Q2.0_ORDERKEY

Output Streams:

7) To Operator #7

Estimated number of rows: 5.0634
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$(A)

Partition Column Names:

+1: Q2.0_ORDERKEY

9) SORT : (Sort)

Cumulative Total Cost: 20.8481
Cumulative CPU Cost: 90472.9
Cumulative I/O Cost: 2
Cumulative Re-Total Cost: 10.4208
Cumulative Re-CPU Cost: 39054.2
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 20.8481
Cumulative Comm Cost: 0
Cumulative First Comm Cost: 0
Estimated Bufferpool Buffers: 3

Arguments:

DUPLWARN: (Duplicates Warning flag)
TRUE

NUMROWS : (Estimated number of rows)
6

ROWWIDTH: (Estimated width of rows)

```

12
SORTKEY : (Sort Key column)
1: Q2.$RID$(A)
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
TRUE

```

Input Streams:

5) From Operator #10

```

Estimated number of rows: 5.0634
Partition Map ID: 4
Partitioning: (MULT )
Multiple Partitions
Number of columns: 2
Subquery predicate ID: Not Applicable

```

Column Names:

+Q2.0_CUSTKEY(A)+Q2.\$RID\$(A)

Partition Column Names:

+1: Q2.0_ORDERKEY

Output Streams:

6) To Operator #8

```

Estimated number of rows: 5.0634
Partition Map ID: 4
Partitioning: (MULT )
Multiple Partitions
Number of columns: 1
Subquery predicate ID: Not Applicable

```

Column Names:

+Q2.\$RID\$(A)

Partition Column Names:

+1: Q2.0_ORDERKEY

10) IXSCAN: (Index Scan)

Cumulative Total Cost: 20.8465
Cumulative CPU Cost: 87515.2
Cumulative I/O Cost: 2
Cumulative Re-Total Cost: 10.4208
Cumulative Re-CPU Cost: 39054.2
Cumulative Re-I/O Cost: 1
Cumulative First Row Cost: 20.8419
Cumulative Comm Cost:0
Cumulative First Comm Cost:0
Estimated Bufferpool Buffers: 3

Arguments:

MAXPAGES: (Maximum pages for prefetch)
1
PREFETCH: (Type of Prefetch)
NONE
ROWLOCK : (Row Lock intent)
NONE
SCANDIR : (Scan Direction)
FORWARD
TABLOCK : (Table Lock intent)
INTENT NONE

Predicates:

2) Start Key Predicate
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 5.06148e-07

Predicate Text:

(Q2.O_CUSTKEY = Q1.C_CUSTKEY)

2) Stop Key Predicate
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 5.06148e-07

Predicate Text:

(Q2.O_CUSTKEY = Q1.C_CUSTKEY)

Input Streams:

4) From Object TPCD.O_CK

Estimated number of rows: 1.00038e+07
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.0_CUSTKEY(A)+Q2.\$RID\$

Partition Column Names:

+1: Q2.0_ORDERKEY

Output Streams:

5) To Operator #9

Estimated number of rows: 5.0634
Partition Map ID: 4
Partitioning: (MULT)
Multiple Partitions
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.0_CUSTKEY(A)+Q2.\$RID\$

Partition Column Names:

+1: Q2.0_ORDERKEY

Objects Used in Access Plan:

Schema: TPCD

Name: O_CK

Type: Index

Time of creation: 2004-06-04-07.02.17.390759

Last statistics update: 2004-06-09-23.20.10.138687

Number of columns: 1

Number of rows: 10003789

Width of rows: -1

Number of buffer pool pages: 296184

Distinct row values: No

Tablespace name: INDEX_TS
 Tablespace overhead: 9.500000
 Tablespace transfer rate: 0.900000
 Source for statistics: Single Node
 Prefetch page count: 48
 Container extent page count: 16
 Index clustering statistic: 0.000046
 Index leaf pages: 18875
 Index tree levels: 3
 Index full key cardinality: 1975707
 Index first key cardinality: 1975707
 Index first 2 keys cardinality: -1
 Index first 3 keys cardinality: -1
 Index first 4 keys cardinality: -1
 Index sequential pages: 18874
 Index page density: 99
 Index avg sequential pages: 18874
 Index avg gap between sequences: 0
 Index avg random pages: 0
 Fetch avg sequential pages: -1
 Fetch avg gap between sequences: -1
 Fetch avg random pages: -1
 Index RID count: 10003789
 Index deleted RID count: 0
 Index empty leaf pages: 0
 Base Table Schema: TPCD
 Base Table Name: ORDERS
 Columns in index:
 0_CUSTKEY

Schema: ORA
 Name: CUSTOMER
 Type: Nickname
 Time of creation: 2004-06-10-08.32.16.950465
 Last statistics update:
 Number of columns: 8
 Number of rows: 1500000
 Width of rows: 236
 Number of buffer pool pages: 32173
 Distinct row values: No
 Tablespace name:
 Tablespace overhead: 24.100000
 Tablespace transfer rate: 0.900000
 Source for statistics: Single Node
 Prefetch page count: 32
 Container extent page count: 32

Schema: TPCD
 Name: ORDERS

Type: Table
Time of creation: 2004-06-04-07.02.15.736633
Last statistics update: 2004-06-09-23.20.10.138687
Number of columns: 9
Number of rows: 10003789
Width of rows: 115
Number of buffer pool pages: 296184
Distinct row values: No
Tablespace name: DATA_TS
Tablespace overhead: 9.500000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 48
Container extent page count: 16
Table overflow record count: 0
Table Active Blocks: -1

Access plan description

The access plan graph for our SQL query has been highlighted in Example B-10 on page 516 under “Access Plan”.

Reading this graph bottom up reveals a significant difference in the access plan for the same query when a fenced wrapper is used:

- ▶ The directed table queue operator (DTQ operator 2) indicates that a nested loop join (NLJOIN operator 3) of the orders table and customer table occurs in parallel on individual partitions, and the combined result of the rows (2.27853e+07) from the individual partitions (7.5951e+06) is directed to the coordinator partition and the RETURN operator 1.
- ▶ As in the previous trusted wrapper scenario, the bottom left part of the graph indicates that the ORA.CUSTOMER nickname has 1.5 million rows according to the DB2 II catalog statistics, and that all these rows are processed by the SHIP operator 5.
- ▶ The BTQ operator 4 represents the distribution of remote data from the fmp process of the fenced wrapper to the db2agntp processes that were created in each of the partitions for accessing the TPCD.ORDERS table.

The 1.5e+06 rows from the SHIP operator 5 will be the outer table (see Arguments of the BTQ operator 4) of a nested loop join (NLJOIN operator 3) with the orders table. The inner table of this nested loop join is estimated to return 5.0634 rows, and the result of the nested loop join is estimated to be 7.5951e+06 rows.
- ▶ The 5.0634 rows for the inner table of the nested loop join are formed from a series of operator executions involving a IXSCAN (operator 10), SORT (operator 9), RIDSCN (operator 8), and FETCH (operator 7). We have not

elaborated on these operators since they follow the same logic discussed in earlier sections.

- ▶ The total cost is estimated to be 214094 timerons, and intra-parallelism is not used (Query Degree:1). However, inter-partition parallelism is used, as indicated by the presence of the BTQ and DTQ operators.

Analysis

Important: Bearing in mind that the EXPLAIN output information is DB2 optimizer estimates only and does not necessarily reflect actual runtime metrics, you should be cautious in drawing conclusions about relative costs associated with each operator, as well as relying on the number of estimated rows retrieved or returned by the various operators. EXPLAIN output information is best used in conjunction with runtime metrics gathered through monitoring.

The following observations apply to the **db2exfmt** output shown in Example B-10 on page 516.

- ▶ DTQ appears above the NLJOIN, meaning that the join is done in the partitions and not by the db2agent process at the coordinator node.
- ▶ BTQ above SHIP means that the data received from the SHIP operation is distributed to all the partitions so that the NLJOIN between the data from the nickname and the ORDERS table can be done in the partitions.
- ▶ The IXSCAN operator indicates that an index on the ORDERS table could be used to find the needed records.
- ▶ The total cost is estimated to be 214094 timerons, which is considerably less than the estimated cost of 3.4124e+07 timerons with the trusted wrapper. Examining the Remote communication cost (1.18291e+06) field in the RETURN operator 1 shows that, while the remote communications cost is the same in both the trusted and fenced wrapper cases, the local processing costs are much higher for the trusted wrapper scenario. The significant difference is the broadcast of nickname data to the individual partitions, which enable the join of the orders and customer table to occur completely in parallel on the federated server.

Note: There is no indication of the wrapper type (trusted or fenced) in the **db2exfmt** output.

The wrapper type is stored in DB2 system catalog table SYSIBM.SYSWRAPOPTIONS or view SYSCAT.WRAPOPTIONS

Our analysis points to the potential for significant performance improvement using the fenced wrapper for this particular SQL query, which should be validated in a regression test environment before committing the change to a production environment.

DB2_MAXIMAL_PUSHDOWN = 'N'

We used the 32-bit instance for this example. Our objective was to review the access plan in a **db2exfmt** output for a federated query that joins nicknames referencing remote DB2 and Oracle data sources with the default server option of DB2_MAXIMAL_PUSHDOWN = 'N', as well as with DB2_MAXIMAL_PUSHDOWN = 'Y'. No local data access is involved, and intra-partition parallelism is disabled (database manager configuration option INTRA_PARALLEL = NO).

Our SQL query finds which supplier should be selected to place an order for a given (brass, size 15) part in a given region (Europe) based on the minimum supplier cost. The parts, supplier, and parts supplier tables reside on DB2, while the nation and region tables reside on Oracle.

Example B-12 shows the complete **db2exfmt** output for our SQL query. The SQL statement we issued has been highlighted under “Original statement” in this output.

Example: B-12 db2exfmt output for DB2_MAXIMAL_PUSHDOWN = 'N'

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

***** EXPLAIN INSTANCE *****

DB2_VERSION: 08.02.0
SOURCE_NAME: SQLC2E05
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-07-01-16.44.30.615209
EXPLAIN_REQUESTER: DB2I32

Database Context:

Parallelism: None
CPU Speed: 4.802167e-07
Comm Speed: 100
Buffer Pool size: 75000

Sort Heap size: 20000
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 1020

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 203 -----

QUERYNO: 1
QUERYTAG:
Statement Type: Select
Udatable: No
Deletable: No
Query Degree: 1

Original Statement:

SELECT S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY, P_MFGR, S_ADDRESS, S_PHONE,
S_COMMENT
FROM DB2.PART, DB2.SUPPLIER, DB2.PARTSUPP, ORA.NATION, ORA.REGION
WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY AND P_SIZE = 15 AND
P_TYPE LIKE '%BRASS' AND S_NATIONKEY = N_NATIONKEY AND R_NAME =
'EUROPE' AND PS_SUPPLYCOST =
(SELECT MIN(PS_SUPPLYCOST)
FROM db2.PARTSUPP, db2.SUPPLIER, ora.NATION, ora.REGION
WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY AND S_NATIONKEY =
N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY AND R_NAME = 'EUROPE')
ORDER BY S_ACCTBAL DESC, N_NAME, S_NAME, P_PARTKEY
FETCH FIRST 100 ROWS ONLY

Optimized Statement:

SELECT Q10.S_ACCTBAL AS "S_ACCTBAL", Q10.S_NAME AS "S_NAME", Q8.N_NAME AS
"N_NAME", Q11.P_PARTKEY AS "P_PARTKEY", Q11.P_MFGR AS "P_MFGR",
Q10.S_ADDRESS AS "S_ADDRESS", Q10.S_PHONE AS "S_PHONE", Q10.S_COMMENT
AS "S_COMMENT"
FROM
(SELECT MIN(Q5.\$C0)
FROM

```

(SELECT Q4.PS_SUPPLYCOST
FROM ORA.REGION AS Q1, ORA.NATION AS Q2, DB2.SUPPLIER AS Q3,
      DB2.PARTSUPP AS Q4
WHERE (Q1.R_NAME = 'EUROPE ') AND (Q2.N_REGIONKEY = Q1.R_REGIONKEY) AND
      (Q3.S_NATIONKEY = Q2.N_NATIONKEY) AND (Q3.S_SUPPKEY =
      Q4.PS_SUPPKEY) AND (Q11.P_PARTKEY = Q4.PS_PARTKEY)) AS Q5) AS
Q6, ORA.REGION AS Q7, ORA.NATION AS Q8, DB2.PARTSUPP AS Q9,
      DB2.SUPPLIER AS Q10, DB2.PART AS Q11
WHERE (Q9.PS_SUPPLYCOST = Q6.$C0) AND (Q7.R_NAME = 'EUROPE ') AND
      (Q10.S_NATIONKEY = Q8.N_NATIONKEY) AND (Q11.P_TYPE LIKE '%BRASS') AND
      (Q11.P_SIZE = 15) AND (Q10.S_SUPPKEY = Q9.PS_SUPPKEY) AND
      (Q11.P_PARTKEY = Q9.PS_PARTKEY)
ORDER BY Q10.S_ACCTBAL DESC, Q8.N_NAME, Q10.S_NAME, Q11.P_PARTKEY

```

Access Plan:

Total Cost: 696494

Query Degree:1

```

Rows
RETURN
( 1)
Cost
I/O
|
0.41902
NLJOIN
( 2)
696494
27710.2
/-----+-----\
0.41902      1
TBSCAN      SHIP
( 3)      ( 33)
696494      0.00418893
27710.2      0
|      |
0.41902      5
SORT      NICKNM: ORA
( 4)      REGION
696494
27710.2
|
0.41902
NLJOIN
( 5)
696494
27710.2
/-----+-----\

```

```

                                0.41902                                1
                                NLJOIN                                SHIP
                                ( 6)                                ( 31)
                                696494                                0.00605841
                                27710.2                                0
                                /-----+-----\
                                0.41902                                1
                                NLJOIN                                SHIP
                                ( 7)                                ( 28)
                                696419                                75.0255
                                27707.2                                3
                                /-----+-----\
                                36042.4                                1.16257e-05
                                SHIP                                FILTER
                                ( 8)                                ( 12)
                                692587                                275.117
                                27696.2                                11
                                /-----+-----\
                                2e+06                                8e+06
                                NICKNM: DB2                                NICKNM: DB2
                                PART                                PARTSUPP
                                GRPBY
                                ( 13)
                                275.116
                                11
                                |
                                1
                                MSJOIN
                                ( 14)
                                275.116
                                11
                                /-----+-----\
                                4                                0.2
                                TBSCAN                                FILTER
                                ( 15)                                ( 21)
                                275.092                                0.022021
                                11                                0
                                |                                |
                                4                                5
                                SORT                                TBSCAN
                                ( 16)                                ( 22)
                                275.089                                0.022021
                                11                                0
                                |                                |
                                4                                5
                                SHIP                                SORT
                                ( 17)                                ( 23)
                                275.086                                0.0189106
                                11                                0
                                /-----+-----\
                                100000                                8e+06
                                5

```

NICKNM: DB2 SUPPLIER	NICKNM: DB2 PARTSUPP	SHIP (24) 0.0160508 0 /-----+\
	5 NICKNM: ORA REGION	25 NICKNM: ORA NATION

1) RETURN: (Return Result)

Cumulative Total Cost: 696494
 Cumulative CPU Cost: 7.78741e+09
 Cumulative I/O Cost: 27710.2
 Cumulative Re-Total Cost: 3656.98
 Cumulative Re-CPU Cost: 7.61527e+09
 Cumulative Re-I/O Cost: 0
 Cumulative First Row Cost: 696494
 Estimated Bufferpool Buffers: 1
 Remote communication cost:20278.7

Arguments:

 BLDLEVEL: (Build level)
 DB2 v8.1.1.64 : s040509
 ENVVAR : (Environment Variable)
 DB2_EXTENDED_OPTIMIZATION = ON
 STMTHEAP: (Statement heap size)
 8192

Input Streams:

29) From Operator #2

Estimated number of rows: 0.41902
 Number of columns: 8
 Subquery predicate ID: Not Applicable

Column Names:

 +Q12.S_ACCTBAL(D)+Q12.N_NAME(A)+Q12.S_NAME(A)
 +Q12.P_PARTKEY(A)+Q12.S_COMMENT+Q12.S_PHONE
 +Q12.S_ADDRESS+Q12.P_MFGR

2) NLJOIN: (Nested Loop Join)

Cumulative Total Cost: 696494

Cumulative CPU Cost: 7.78741e+09
Cumulative I/O Cost: 27710.2
Cumulative Re-Total Cost: 3656.98
Cumulative Re-CPU Cost: 7.61527e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 696494
Estimated Bufferpool Buffers: 1
Remote communication cost:20278.7

Arguments:

EARLYOUT: (Early Out flag)
 NONE
FETCHMAX: (Override for FETCH MAXPAGES)
 IGNORE
ISCANMAX: (Override for ISCAN MAXPAGES)
 IGNORE

Input Streams:

26) From Operator #3

 Estimated number of rows: 0.41902
 Number of columns: 8
 Subquery predicate ID: Not Applicable

 Column Names:

 +Q10.S_ACCTBAL(D)+Q8.N_NAME(A)+Q10.S_NAME(A)
 +Q11.P_PARTKEY(A)+Q10.S_COMMENT+Q10.S_PHONE
 +Q10.S_ADDRESS+Q11.P_MFGR

28) From Operator #33

 Estimated number of rows: 1
 Number of columns: 0
 Subquery predicate ID: Not Applicable

Output Streams:

29) To Operator #1

 Estimated number of rows: 0.41902
 Number of columns: 8
 Subquery predicate ID: Not Applicable

 Column Names:

```
+Q12.S_ACCTBAL(D)+Q12.N_NAME(A)+Q12.S_NAME(A)
+Q12.P_PARTKEY(A)+Q12.S_COMMENT+Q12.S_PHONE
+Q12.S_ADDRESS+Q12.P_MFGR
```

3) TBSCAN: (Table Scan)

```
Cumulative Total Cost: 696494
Cumulative CPU Cost: 7.7874e+09
Cumulative I/O Cost: 27710.2
Cumulative Re-Total Cost: 3656.98
Cumulative Re-CPU Cost: 7.61527e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 696494
Estimated Bufferpool Buffers: 0
Remote communication cost:20272.5
```

Arguments:

```
-----
JN INPUT: (Join input leg)
  OUTER
MAXPAGES: (Maximum pages for prefetch)
  ALL
PREFETCH: (Type of Prefetch)
  NONE
SCANDIR : (Scan Direction)
  FORWARD
```

Input Streams:

25) From Operator #4

```
Estimated number of rows: 0.41902
Number of columns: 8
Subquery predicate ID: Not Applicable
```

Column Names:

```
+Q10.S_ACCTBAL(D)+Q8.N_NAME(A)+Q10.S_NAME(A)
+Q11.P_PARTKEY(A)+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q11.P_MFGR
```

Output Streams:

26) To Operator #2

```
Estimated number of rows: 0.41902
Number of columns: 8
Subquery predicate ID: Not Applicable
```

Column Names:

+Q10.S_ACCTBAL(D)+Q8.N_NAME(A)+Q10.S_NAME(A)
+Q11.P_PARTKEY(A)+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q11.P_MFGR

4) SORT : (Sort)

Cumulative Total Cost: 696494
Cumulative CPU Cost: 7.7874e+09
Cumulative I/O Cost: 27710.2
Cumulative Re-Total Cost: 3656.98
Cumulative Re-CPU Cost: 7.61527e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 696494
Estimated Bufferpool Buffers: 27701.2
Remote communication cost:20272.5

Arguments:

DUPLWARN: (Duplicates Warning flag)
FALSE
NUMROWS : (Estimated number of rows)
1
ROWWIDTH: (Estimated width of rows)
204
SORTKEY : (Sort Key column)
1: Q10.S_ACCTBAL(D)
SORTKEY : (Sort Key column)
2: Q8.N_NAME(A)
SORTKEY : (Sort Key column)
3: Q10.S_NAME(A)
SORTKEY : (Sort Key column)
4: Q11.P_PARTKEY(A)
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

24) From Operator #5

Estimated number of rows: 0.41902
Number of columns: 12
Subquery predicate ID: Not Applicable

Column Names:

```

-----
+Q6.$C0+Q8.N_NAME+Q9.PS_SUPPLYCOST
+Q9.PS_SUPPKEY+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q10.S_NAME+Q10.S_ACCTBAL
+Q10.S_NATIONKEY+Q11.P_MFGR+Q11.P_PARTKEY

```

Output Streams:

25) To Operator #3

```

Estimated number of rows: 0.41902
Number of columns: 8
Subquery predicate ID: Not Applicable

```

Column Names:

```

-----
+Q10.S_ACCTBAL(D)+Q8.N_NAME(A)+Q10.S_NAME(A)
+Q11.P_PARTKEY(A)+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q11.P_MFGR

```

5) NLJOIN: (Nested Loop Join)

```

Cumulative Total Cost: 696494
Cumulative CPU Cost: 7.78739e+09
Cumulative I/O Cost: 27710.2
Cumulative Re-Total Cost: 3656.98
Cumulative Re-CPU Cost: 7.61527e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 696494
Estimated Bufferpool Buffers: 27701.2
Remote communication cost:20272.5

```

Arguments:

```

-----
EARLYOUT: (Early Out flag)
      NONE
FETCHMAX: (Override for FETCH MAXPAGES)
      IGNORE
ISCANMAX: (Override for ISCAN MAXPAGES)
      IGNORE

```

Predicates:

```

-----
4) Predicate used in Join
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 0.04

```


Predicate Text:

(Q10.S_NATIONKEY = Q8.N_NATIONKEY)

Input Streams:

21) From Operator #6

Estimated number of rows: 0.41902

Number of columns: 11

Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0+Q9.PS_SUPPLYCOST+Q9.PS_SUPPKEY
+Q10.S_COMMENT+Q10.S_PHONE+Q10.S_ADDRESS
+Q10.S_NAME+Q10.S_ACCTBAL+Q10.S_NATIONKEY
+Q11.P_MFGR+Q11.P_PARTKEY

23) From Operator #31

Estimated number of rows: 1

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q8.N_NAME

Output Streams:

24) To Operator #4

Estimated number of rows: 0.41902

Number of columns: 12

Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0+Q8.N_NAME+Q9.PS_SUPPLYCOST
+Q9.PS_SUPPKEY+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q10.S_NAME+Q10.S_ACCTBAL
+Q10.S_NATIONKEY+Q11.P_MFGR+Q11.P_PARTKEY

6) NLJOIN: (Nested Loop Join)

Cumulative Total Cost: 696494

Cumulative CPU Cost: 7.78738e+09
Cumulative I/O Cost: 27710.2
Cumulative Re-Total Cost: 3656.97
Cumulative Re-CPU Cost: 7.61526e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 696494
Estimated Bufferpool Buffers: 27700.2
Remote communication cost:20266.3

Arguments:

EARLYOUT: (Early Out flag)
 NONE
FETCHMAX: (Override for FETCH MAXPAGES)
 IGNORE
ISCANMAX: (Override for ISCAN MAXPAGES)
 IGNORE
JN INPUT: (Join input leg)
 OUTER

Predicates:

7) Predicate used in Join
 Relational Operator: Equal (=)
 Subquery Input Required: No
 Filter Factor: 1e-05

Predicate Text:

(Q10.S_SUPPKEY = Q9.PS_SUPPKEY)

Input Streams:

18) From Operator #7

 Estimated number of rows: 0.41902
 Number of columns: 5
 Subquery predicate ID: Not Applicable

 Column Names:

 +Q6.\$C0+Q9.PS_SUPPLYCOST+Q9.PS_SUPPKEY
 +Q11.P_MFGR+Q11.P_PARTKEY

20) From Operator #28

 Estimated number of rows: 1
 Number of columns: 7

Subquery predicate ID: Not Applicable

Column Names:

+Q10.S_SUPPKEY(A)+Q10.S_NATIONKEY(A)
+Q10.S_COMMENT+Q10.S_PHONE+Q10.S_ADDRESS
+Q10.S_NAME+Q10.S_ACCTBAL

Output Streams:

21) To Operator #5

Estimated number of rows: 0.41902

Number of columns: 11

Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0+Q9.PS_SUPPLYCOST+Q9.PS_SUPPKEY
+Q10.S_COMMENT+Q10.S_PHONE+Q10.S_ADDRESS
+Q10.S_NAME+Q10.S_ACCTBAL+Q10.S_NATIONKEY
+Q11.P_MFGR+Q11.P_PARTKEY

7) NLJOIN: (Nested Loop Join)

Cumulative Total Cost: 696419

Cumulative CPU Cost: 7.78732e+09

Cumulative I/O Cost: 27707.2

Cumulative Re-Total Cost: 3656.96

Cumulative Re-CPU Cost: 7.61523e+09

Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 696419

Estimated Bufferpool Buffers: 27697.2

Remote communication cost:20260.1

Arguments:

EARLYOUT: (Early Out flag)

NONE

FETCHMAX: (Override for FETCH MAXPAGES)

IGNORE

ISCANMAX: (Override for ISCAN MAXPAGES)

IGNORE

JN INPUT: (Join input leg)

OUTER

Predicates:

2) Predicate used in Join
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 1.16257e-05

Predicate Text:

(Q9.PS_SUPPLYCOST = Q6.\$C0)

Input Streams:

3) From Operator #8

Estimated number of rows: 36042.4
Number of columns: 4
Subquery predicate ID: Not Applicable

Column Names:

+Q9.PS_SUPPLYCOST+Q9.PS_SUPPKEY+Q11.P_MFGR
+Q11.P_PARTKEY

17) From Operator #12

Estimated number of rows: 1.16257e-05
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0

Output Streams:

18) To Operator #6

Estimated number of rows: 0.41902
Number of columns: 5
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0+Q9.PS_SUPPLYCOST+Q9.PS_SUPPKEY
+Q11.P_MFGR+Q11.P_PARTKEY

8) SHIP : (Ship)

Cumulative Total Cost: 692587
Cumulative CPU Cost: 3.79842e+08
Cumulative I/O Cost: 27696.2
Cumulative Re-Total Cost: 99.7818
Cumulative Re-CPU Cost: 2.07785e+08
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 151.477
Estimated Bufferpool Buffers: 27697.2
Remote communication cost:20237.9

Arguments:

CSERQY : (Remote common subexpression)
FALSE
DSTSEVER: (Destination (ship to) server)
- (NULL).
JN INPUT: (Join input leg)
OUTER
RMTQTX : (Remote statement)
SELECT A0."P_PARTKEY", A0."P_MFGR", A1."PS_SUPPKEY",
A1."PS_SUPPLYCOST" FROM "TPCD"."PART" A0, "TPCD"."PARTSUPP" A1 WHERE
(A0."P_SIZE" = 15) AND (A0."P_TYPE" LIKE '%BRASS') AND (A0."P_PARTKEY" =
A1."PS_PARTKEY") FOR READ ONLY
SRCSEVER: (Source (ship from) server)
DB2SERV
STREAM : (Remote stream)
FALSE

Input Streams:

1) From Object DB2.PART

Estimated number of rows: 2e+06
Number of columns: 5
Subquery predicate ID: Not Applicable

Column Names:

+Q11.\$RID\$+Q11.P_MFGR+Q11.P_TYPE+Q11.P_SIZE
+Q11.P_PARTKEY

2) From Object DB2.PARTSUPP

Estimated number of rows: 8e+06
Number of columns: 4
Subquery predicate ID: Not Applicable

Column Names:

+Q9.\$RID\$+Q9.PS_SUPPLYCOST+Q9.PS_SUPPKEY
+Q9.PS_PARTKEY

Output Streams:

3) To Operator #7

Estimated number of rows: 36042.4
Number of columns: 4
Subquery predicate ID: Not Applicable

Column Names:

+Q9.PS_SUPPLYCOST+Q9.PS_SUPPKEY+Q11.P_MFGR
+Q11.P_PARTKEY

12) FILTER: (Filter)

Cumulative Total Cost: 275.117
Cumulative CPU Cost: 242920
Cumulative I/O Cost: 11
Cumulative Re-Total Cost: 275.117
Cumulative Re-CPU Cost: 242920
Cumulative Re-I/O Cost: 11
Cumulative First Row Cost: 275.117
Estimated Bufferpool Buffers: 0
Remote communication cost:22.2188

Arguments:

JN INPUT: (Join input leg)
INNER

Predicates:

2) Residual Predicate
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 1.16257e-05

Predicate Text:

(Q9.PS_SUPPLYCOST = Q6.\$C0)

Input Streams:

16) From Operator #13

Estimated number of rows: 1
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0

Output Streams:

17) To Operator #7

Estimated number of rows: 1.16257e-05
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0

13) GRPBY : (Group By)

Cumulative Total Cost: 275.116

Cumulative CPU Cost: 241585

Cumulative I/O Cost: 11

Cumulative Re-Total Cost: 275.116

Cumulative Re-CPU Cost: 241585

Cumulative Re-I/O Cost: 11

Cumulative First Row Cost: 275.116

Estimated Bufferpool Buffers: 0

Remote communication cost:22.2188

Arguments:

AGGMODE : (Aggregation Mode)

COMPLETE

GROUPBYC: (Group By columns)

FALSE

GROUPBYN: (Number of Group By columns)

0

ONEFETCH: (One Fetch flag)

FALSE

Input Streams:

15) From Operator #14

Estimated number of rows: 0.8
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q5.PS_SUPPLYCOST

Output Streams:

16) To Operator #12

Estimated number of rows: 1
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0

14) MSJOIN: (Merge Scan Join)

Cumulative Total Cost: 275.116
Cumulative CPU Cost: 241135
Cumulative I/O Cost: 11
Cumulative Re-Total Cost: 275.116
Cumulative Re-CPU Cost: 241135
Cumulative Re-I/O Cost: 11
Cumulative First Row Cost: 275.116
Estimated Bufferpool Buffers: 0
Remote communication cost:22.2188

Arguments:

EARLYOUT: (Early Out flag)

NONE

INNERCOL: (Inner Order Columns)

1: Q2.N_NATIONKEY(A)

OUTERCOL: (Outer Order columns)

1: Q3.S_NATIONKEY(A)

TEMPSIZE: (Temporary Table Page Size)

4096

Predicates:

11) Predicate used in Join

Relational Operator: Equal (=)

Subquery Input Required: No

Filter Factor: 0.04

Predicate Text:

(Q3.S_NATIONKEY = Q2.N_NATIONKEY)

Input Streams:

8) From Operator #15

Estimated number of rows: 4

Number of columns: 2

Subquery predicate ID: Not Applicable

Column Names:

+Q3.S_NATIONKEY(A)+Q4.PS_SUPPLYCOST

14) From Operator #21

Estimated number of rows: 0.2

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

Output Streams:

15) To Operator #13

Estimated number of rows: 0.8

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q5.PS_SUPPLYCOST

15) TBSCAN: (Table Scan)

Cumulative Total Cost: 275.092

Cumulative CPU Cost: 191030

Cumulative I/O Cost: 11

Cumulative Re-Total Cost: 25.0504

Cumulative Re-CPU Cost: 104891

Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 275.09
Estimated Bufferpool Buffers: 0
Remote communication cost:10.8594

Arguments:

JN INPUT: (Join input leg)
OUTER
MAXPAGES: (Maximum pages for prefetch)
ALL
PREFETCH: (Type of Prefetch)
NONE
SCANDIR : (Scan Direction)
FORWARD

Input Streams:

7) From Operator #16

Estimated number of rows: 4
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q3.S_NATIONKEY(A)+Q4.PS_SUPPLYCOST

Output Streams:

8) To Operator #14

Estimated number of rows: 4
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q3.S_NATIONKEY(A)+Q4.PS_SUPPLYCOST

16) SORT : (Sort)
Cumulative Total Cost: 275.089
Cumulative CPU Cost: 185333
Cumulative I/O Cost: 11
Cumulative Re-Total Cost: 25.0476
Cumulative Re-CPU Cost: 99194
Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 275.089
Estimated Bufferpool Buffers: 50012
Remote communication cost:10.8594

Arguments:

DUPLWARN: (Duplicates Warning flag)
FALSE
NUMROWS : (Estimated number of rows)
4
ROWWIDTH: (Estimated width of rows)
16
SORTKEY : (Sort Key column)
1: Q3.S_NATIONKEY(A)
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

6) From Operator #17

Estimated number of rows: 4
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q3.S_NATIONKEY+Q4.PS_SUPPLYCOST

Output Streams:

7) To Operator #15

Estimated number of rows: 4
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q3.S_NATIONKEY(A)+Q4.PS_SUPPLYCOST

17) SHIP : (Ship)
Cumulative Total Cost: 275.086
Cumulative CPU Cost: 180037
Cumulative I/O Cost: 11

Cumulative Re-Total Cost: 25.0476
Cumulative Re-CPU Cost: 99194
Cumulative Re-I/O Cost: 1
Cumulative First Row Cost: 125.044
Estimated Bufferpool Buffers: 50012
Remote communication cost:10.8594

Arguments:

CSERQY : (Remote common subexpression)
FALSE
DSTSEVER: (Destination (ship to) server)
- (NULL).
RMTQTX : (Remote statement)
SELECT A0."PS_SUPPLYCOST", A1."S_NATIONKEY" FROM "TPCD"."PARTSUPP"
A0, "TPCD"."SUPPLIER" A1 WHERE (:HO = A0."PS_PARTKEY") AND (A1."S_SUPPKEY" =
A0."PS_SUPPKEY") FOR READ ONLY
SRCSEVER: (Source (ship from) server)
DB2SERV
STREAM : (Remote stream)
FALSE

Input Streams:

4) From Object DB2.PARTSUPP

Estimated number of rows: 8e+06
Number of columns: 4
Subquery predicate ID: Not Applicable

Column Names:

+Q4.\$RID\$+Q4.PS_SUPPLYCOST+Q4.PS_SUPPKEY
+Q4.PS_PARTKEY

5) From Object DB2.SUPPLIER

Estimated number of rows: 100000
Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:

+Q3.\$RID\$+Q3.S_NATIONKEY+Q3.S_SUPPKEY

Output Streams:

6) To Operator #16

Estimated number of rows: 4
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q3.S_NATIONKEY+Q4.PS_SUPPLYCOST

21) FILTER: (Filter)

Cumulative Total Cost: 0.022021
Cumulative CPU Cost: 45856.4
Cumulative I/O Cost: 0
Cumulative Re-Total Cost: 0.00924657
Cumulative Re-CPU Cost: 19255
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 0.0196007
Estimated Bufferpool Buffers: 0
Remote communication cost:11.3594

Arguments:

JN INPUT: (Join input leg)
INNER

Predicates:

11) Residual Predicate
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 0.04

Predicate Text:

(Q3.S_NATIONKEY = Q2.N_NATIONKEY)

Input Streams:

13) From Operator #22

Estimated number of rows: 5
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

Output Streams:

14) To Operator #14

Estimated number of rows: 0.2

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

22) TBSCAN: (Table Scan)

Cumulative Total Cost: 0.022021

Cumulative CPU Cost: 45856.4

Cumulative I/O Cost: 0

Cumulative Re-Total Cost: 0.00924657

Cumulative Re-CPU Cost: 19255

Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 0.0196007

Estimated Bufferpool Buffers: 0

Remote communication cost:11.3594

Arguments:

MAXPAGES: (Maximum pages for prefetch)

ALL

PREFETCH: (Type of Prefetch)

NONE

SCANDIR : (Scan Direction)

FORWARD

Input Streams:

12) From Operator #23

Estimated number of rows: 5

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

Output Streams:

13) To Operator #21

Estimated number of rows: 5
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

23) SORT : (Sort)

Cumulative Total Cost: 0.0189106
Cumulative CPU Cost: 39379.4
Cumulative I/O Cost: 0
Cumulative Re-Total Cost: 0.00613621
Cumulative Re-CPU Cost: 12778
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 0.0189106
Estimated Bufferpool Buffers: 2
Remote communication cost:11.3594

Arguments:

DUPLWARN: (Duplicates Warning flag)
FALSE
NUMROWS : (Estimated number of rows)
5
ROWWIDTH: (Estimated width of rows)
8
SORTKEY : (Sort Key column)
1: Q2.N_NATIONKEY(A)
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

11) From Operator #24

Estimated number of rows: 5
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY

Output Streams:

12) To Operator #22

Estimated number of rows: 5

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

24) SHIP : (Ship)

Cumulative Total Cost: 0.0160508

Cumulative CPU Cost: 33424

Cumulative I/O Cost: 0

Cumulative Re-Total Cost: 0.00613621

Cumulative Re-CPU Cost: 12778

Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 0.00919759

Estimated Bufferpool Buffers: 2

Remote communication cost:11.3594

Arguments:

CSERQY : (Remote common subexpression)

FALSE

DSTSEVER: (Destination (ship to) server)

- (NULL).

RMTQTX : (Remote statement)

SELECT A1."N_NATIONKEY" FROM "IITEST"."REGION" A0, "IITEST"."NATION"

A1 WHERE (A0."R_NAME" = 'EUROPE ') AND (A1."N_REGIONKEY" =

A0."R_REGIONKEY")

SRCSEVER: (Source (ship from) server)

ORASERV

STREAM : (Remote stream)

FALSE

Input Streams:

9) From Object ORA.REGION

Estimated number of rows: 5

Number of columns: 3

Subquery predicate ID: Not Applicable

Column Names:

+Q1.\$RID\$+Q1.R_NAME+Q1.R_REGIONKEY

10) From Object ORA.NATION

Estimated number of rows: 25

Number of columns: 3

Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$+Q2.N_REGIONKEY+Q2.N_NATIONKEY

Output Streams:

11) To Operator #23

Estimated number of rows: 5

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY

28) SHIP : (Ship)

Cumulative Total Cost: 75.0255

Cumulative CPU Cost: 53193.6

Cumulative I/O Cost: 3

Cumulative Re-Total Cost: 50.0187

Cumulative Re-CPU Cost: 38888.6

Cumulative Re-I/O Cost: 2

Cumulative First Row Cost: 75.0241

Estimated Bufferpool Buffers: 4520

Remote communication cost:9.35938

Arguments:

CSERQY : (Remote common subexpression)

FALSE

DSTSEVER: (Destination (ship to) server)

- (NULL).

JN INPUT: (Join input leg)

INNER

RMTQTX : (Remote statement)

```

        SELECT AO."S_NATIONKEY", AO."S_ACCTBAL", AO."S_NAME", AO."S_ADDRESS",
        AO."S_PHONE", AO."S_COMMENT" FROM "TPCD"."SUPPLIER" AO WHERE (AO."S_SUPPKEY" =
:HO ) ORDER BY AO."S_SUPPKEY" ASC, 1 ASC FOR READ ONLY
SRCSEVER: (Source (ship from) server)
DB2SERV
STREAM : (Remote stream)
FALSE

```

Input Streams:

19) From Object DB2.SUPPLIER

```

Estimated number of rows: 100000
Number of columns: 5
Subquery predicate ID: Not Applicable

```

Column Names:

```

+Q10.S_COMMENT+Q10.S_PHONE+Q10.S_ADDRESS
+Q10.S_NAME+Q10.S_ACCTBAL

```

Output Streams:

20) To Operator #6

```

Estimated number of rows: 1
Number of columns: 7
Subquery predicate ID: Not Applicable

```

Column Names:

```

+Q10.S_SUPPKEY(A)+Q10.S_NATIONKEY(A)
+Q10.S_COMMENT+Q10.S_PHONE+Q10.S_ADDRESS
+Q10.S_NAME+Q10.S_ACCTBAL

```

31) SHIP : (Ship)

```

Cumulative Total Cost: 0.00605841
Cumulative CPU Cost: 12616
Cumulative I/O Cost: 0
Cumulative Re-Total Cost: 0.00399012
Cumulative Re-CPU Cost: 8309
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 0.00498321
Estimated Bufferpool Buffers: 1
Remote communication cost:9.35938

```

Arguments:

```

-----
CSERQY : (Remote common subexpression)
  FALSE
DSTSEVER: (Destination (ship to) server)
  - (NULL).
JN INPUT: (Join input leg)
  INNER
RMTQTX : (Remote statement)
  SELECT AO."N_NAME" FROM "IITEST"."NATION" AO WHERE (:HO =
AO."N_NATIONKEY")
SRCSEVER: (Source (ship from) server)
  ORASERV
STREAM : (Remote stream)
  FALSE

```

Input Streams:

```

-----
22) From Object ORA.NATION

```

```

  Estimated number of rows: 25
  Number of columns: 3
  Subquery predicate ID: Not Applicable

```

```

  Column Names:
  -----

```

```

  +Q8.$RID$+Q8.N_NAME+Q8.N_NATIONKEY

```

Output Streams:

```

-----
23) To Operator #5

```

```

  Estimated number of rows: 1
  Number of columns: 1
  Subquery predicate ID: Not Applicable

```

```

  Column Names:
  -----

```

```

  +Q8.N_NAME

```

```

33) SHIP : (Ship)
  Cumulative Total Cost: 0.00418893
  Cumulative CPU Cost: 8723
  Cumulative I/O Cost: 0
  Cumulative Re-Total Cost: 0.00214609
  Cumulative Re-CPU Cost: 4469
  Cumulative Re-I/O Cost: 0
  Cumulative First Row Cost: 0.00313918

```

Estimated Bufferpool Buffers: 1
Remote communication cost:9.35938

Arguments:

CSERQY : (Remote common subexpression)
FALSE
DSTSEVER: (Destination (ship to) server)
- (NULL).
JN INPUT: (Join input leg)
INNER
RMTQTX : (Remote statement)
SELECT '1' FROM "IITEST"."REGION" AO WHERE (AO."R_NAME" = 'EUROPE
)
SRCSEVER: (Source (ship from) server)
ORASERV
STREAM : (Remote stream)
FALSE

Input Streams:

27) From Object ORA.REGION

Estimated number of rows: 5
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q7.\$RID\$+Q7.R_NAME

Output Streams:

28) To Operator #2

Estimated number of rows: 1
Number of columns: 0
Subquery predicate ID: Not Applicable

Objects Used in Access Plan:

Schema: DB2
Name: PART
Type: Nickname
Time of creation: 2004-06-10-08.38.47.722759
Last statistics update:

Number of columns: 9
Number of rows: 2000000
Width of rows: 70
Number of buffer pool pages: 76238
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: DB2

Name: PARTSUPP

Type: Nickname

Time of creation: 2004-06-10-08.38.47.872025
Last statistics update:
Number of columns: 5
Number of rows: 8000000
Width of rows: 28
Number of buffer pool pages: 319290
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: DB2

Name: SUPPLIER

Type: Nickname

Time of creation: 2004-06-10-08.38.47.795922
Last statistics update:
Number of columns: 7
Number of rows: 100000
Width of rows: 157
Number of buffer pool pages: 4093
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: ORA

Name: NATION

Type: Nickname

Time of creation: 2004-06-10-08.38.38.710626
Last statistics update: 2004-06-10-19.56.42.522824
Number of columns: 4
Number of rows: 25
Width of rows: 57
Number of buffer pool pages: 2
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: ORA

Name: REGION

Type: Nickname

Time of creation: 2004-06-10-08.38.38.810764
Last statistics update: 2004-06-10-19.56.42.820988
Number of columns: 3
Number of rows: 5
Width of rows: 53
Number of buffer pool pages: 2
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Access plan description

The access plan graph for our SQL query has been highlighted in Example B-12 on page 554 under “Access Plan”.

Reading this graph bottom up reveals the following:

- The bottom left part of the graph indicates that a join of the PART and PARTSUPP nicknames with cardinalities of 2e+06 and 8e+06 rows, respectively, according to the DB2 II catalog, are pushed down to the remote DB2 via the SHIP operator 8. The RMTQTX field of Arguments contains the SQL fragment executed at the remote DB2, and returns an estimated 36042.4 rows to the federated server for participating as the outer table (see the JN INPUT field in Arguments of the SHIP operator 8) in a nested loop join (NLJOIN operator 7). Four columns are returned as described in the Output Stream section of the SHIP operator 8.

- ▶ The 1.16257e-05 rows for the inner table of the nested loop join (NLJOIN operator 7) are formed from a series of executions involving SHIP operator 17, SORT operator 16, TBSCAN operator 15, MSJOIN operator 14, SHIP operator 24, SORT operator 23, TBSCAN operator 22, FILTER operator 21, GRPBY operator 13 and FILTER operator 12. The nicknames involved in deriving the rows of the inner table are SUPPLIER and PARTSUPP from DB2, and REGION and NATION from Oracle.
- ▶ The 0.41902 estimated rows from the nested loop join (NLJOIN operator 7) form the outer table of another nested loop join (NLJOIN operator 6) with an inner table that returns an estimated 1 row from the SHIP operator 27, which accesses the SUPPLIER nickname in DB2.
- ▶ The estimated 0.41902 estimated rows from the nested loop join (NLJOIN operator 5) in turn form the outer table of another nested loop join (NLJOIN operator 6) with an inner table that returns an estimated 1 row from the SHIP operator 31, which accesses the NATION nickname in Oracle.
- ▶ The estimated 0.41902 estimated rows from the nested loop join (NLJOIN operator 6) is sorted (SORT operator 4) on four columns (descending S_ACCTBAL, ascending N_NAME, ascending S_NAME, and ascending P_PARTKEY), and written to a temporary table.
- ▶ The TBSCAN operator 3 retrieves the estimated 0.41902 sorted rows in the temporary table, which forms the outer table of another nested loop join (NLJOIN operator 2) with an inner table that returns an estimated 1 row from the SHIP operator 33, which accesses the REGION nickname in Oracle.
- ▶ The 0.41902 estimated rows in the nested loop join result (NLJOIN operator 2) will be returned to the user via the RETURN operator 1.
- ▶ The total cost is estimated to be 696494 timerons, and there is no parallelism involved (Query Degree:1 and no table queue operators in the Access Plan).

Analysis

Important: Bearing in mind that the EXPLAIN output information is DB2 optimizer estimates only and does not necessarily reflect actual runtime metrics, you should be cautious in drawing conclusions about relative costs associated with each operator, as well as relying on the number of estimated rows retrieved or returned by the various operators. EXPLAIN output information is best used in conjunction with runtime metrics gathered through monitoring.

The following observations apply to the **db2exfmt** output shown in Example B-12 on page 554.

- ▶ The cardinality of the nicknames appear to be valid since the DB2 optimizer default of 1000 rows does not appear. This, however, does not guarantee that the DB2 II catalog reflects the actual number of rows at the remote data source.
- ▶ There are a few sorts involved (operators 4, 16 and 23) with very few rows estimated to be sorted. The Database Context shows the Sort Heap size to be 20000 pages, while the Buffer Pool size is 75000. There appear to be no spillover concerns associated with sorting.
- ▶ For some reason, the DB2 optimizer has not chosen to push down the join of the SUPPLIER table (SHIP operator 28) along with the PART and PARTSUPP nicknames (SHIP operator 8), and instead chose to perform an additional join (NLJOIN operator 6).

Our analysis points to a possible investigation of why a 3-way join of PART, PARTSUPP, and SUPPLIER was not pushed down to the remote DB2 data source. However, this further investigation should be conducted only after actual runtime metrics are gathered for this query and performance is considered to be inadequate.

Note: There is no indication of the server option (DB2_MAXIMAL_PUSHDOWN set to 'Y' or 'N') in the **db2exfmt** output.

The server options are stored in the DB2 system catalog table SYSIBM.SYSSERVEROPTIONS or view SYSCAT.SERVEROPTIONS.

DB2_MAXIMAL_PUSHDOWN = 'Y'

In this scenario, we ran the same query and environment used in "DB2_MAXIMAL_PUSHDOWN = 'N'" on page 554, with the only difference being that the server definition was set to DB2_MAXIMAL_PUSHDOWN = 'Y'.

Example B-13 shows the complete **db2exfmt** output for our SQL query. The SQL statement we issued has been highlighted under "Original statement" in this output.

Example: B-13 db2exfmt output for DB2_MAXIMAL_PUSHDOWN = 'Y'

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

***** EXPLAIN INSTANCE *****

DB2_VERSION: 08.02.0
SOURCE_NAME: SQLC2E05
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-07-01-17.24.10.365830
EXPLAIN_REQUESTER: DB2I32

Database Context:

Parallelism: None
CPU Speed: 4.802167e-07
Comm Speed: 100
Buffer Pool size: 75000
Sort Heap size: 20000
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 1020

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 203 -----

QUERYNO: 1
QUERYTAG:
Statement Type: Select
Updatable: No
Deletable: No
Query Degree: 1

Original Statement:

```
SELECT S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY, P_MFGR, S_ADDRESS, S_PHONE,
       S_COMMENT
FROM DB2.PART, DB2.SUPPLIER, DB2.PARTSUPP, ORA.NATION, ORA.REGION
WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY AND P_SIZE = 15 AND
      P_TYPE LIKE '%BRASS' AND S_NATIONKEY = N_NATIONKEY AND R_NAME =
      'EUROPE' AND PS_SUPPLYCOST =
      (SELECT MIN(PS_SUPPLYCOST)
```

```
FROM db2.PARTSUPP, db2.SUPPLIER, ora.NATION, ora.REGION
WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY AND S_NATIONKEY =
      N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY AND R_NAME = 'EUROPE' )
ORDER BY S_ACCTBAL DESC, N_NAME, S_NAME, P_PARTKEY
FETCH FIRST 100 ROWS ONLY
```

Optimized Statement:

```
-----
SELECT Q10.S_ACCTBAL AS "S_ACCTBAL", Q10.S_NAME AS "S_NAME", Q8.N_NAME AS
      "N_NAME", Q11.P_PARTKEY AS "P_PARTKEY", Q11.P_MFGR AS "P_MFGR",
      Q10.S_ADDRESS AS "S_ADDRESS", Q10.S_PHONE AS "S_PHONE", Q10.S_COMMENT
      AS "S_COMMENT"
FROM
      (SELECT MIN(Q5.$C0)
      FROM
            (SELECT Q4.PS_SUPPLYCOST
            FROM ORA.REGION AS Q1, ORA.NATION AS Q2, DB2.SUPPLIER AS Q3,
            DB2.PARTSUPP AS Q4
            WHERE (Q1.R_NAME = 'EUROPE ') AND (Q2.N_REGIONKEY = Q1.R_REGIONKEY) AND
            (Q3.S_NATIONKEY = Q2.N_NATIONKEY) AND (Q3.S_SUPPKEY =
            Q4.PS_SUPPKEY) AND (Q11.P_PARTKEY = Q4.PS_PARTKEY)) AS Q5) AS
      Q6, ORA.REGION AS Q7, ORA.NATION AS Q8, DB2.PARTSUPP AS Q9,
      DB2.SUPPLIER AS Q10, DB2.PART AS Q11
WHERE (Q9.PS_SUPPLYCOST = Q6.$C0) AND (Q7.R_NAME = 'EUROPE ') AND
      (Q10.S_NATIONKEY = Q8.N_NATIONKEY) AND (Q11.P_TYPE LIKE '%BRASS') AND
      (Q11.P_SIZE = 15) AND (Q10.S_SUPPKEY = Q9.PS_SUPPKEY) AND
      (Q11.P_PARTKEY = Q9.PS_PARTKEY)
ORDER BY Q10.S_ACCTBAL DESC, Q8.N_NAME, Q10.S_NAME, Q11.P_PARTKEY
```

Access Plan:

```
-----
Total Cost: 712543
Query Degree:1
```

Rows	
RETURN	
(1)	
Cost	
I/O	
0.41902	
NLJOIN	
(2)	
712543	
31800.2	
/-----\	
0.41902	1
TBSCAN	SHIP

```

( 3)      ( 31)
712543    0.00418893
31800.2    0
|          |
0.41902    5
SORT      NICKNM: ORA
( 4)      REGION
712543
31800.2
|
0.41902
NLJOIN
( 5)
712543
31800.2
/-----+-----\
0.41902    1
NLJOIN    SHIP
( 6)      ( 29)
712543    0.00605841
31800.2    0
|
/-----+-----\
36042.4    1.16257e-05
SHIP      FILTER      NICKNM: ORA
( 7)      ( 13)      NATION
708711    275.117
31789.2    11
|
+-----+-----+
100000    2e+06    8e+06
NICKNM: DB2  NICKNM: DB2  NICKNM: DB2
SUPPLIER    PART    PARTSUPP
|          |
1          1
GRPBY
( 14)
275.116
11
|
0.8
MSJOIN
( 15)
275.116
11
/-----+-----\
4          0.2
TBSCAN    FILTER
( 16)      ( 22)
275.092    0.022021
11          0
|          |
4          5
SORT      TBSCAN
( 17)      ( 23)

```

	275.089	0.022021
	11	0
	4	5
	SHIP	SORT
	(18)	(24)
	275.086	0.0189106
	11	0
	/-----+-----\	
100000	8e+06	5
NICKNM: DB2	NICKNM: DB2	SHIP
SUPPLIER	PARTSUPP	(25)
		0.0160508
		0
		/-----+-----\
		5
25		NICKNM: ORA
ORA		NICKNM:
		REGION
NATION		

1) RETURN: (Return Result)

Cumulative Total Cost: 712543

Cumulative CPU Cost: 7.99833e+09

Cumulative I/O Cost: 31800.2

Cumulative Re-Total Cost: 712268

Cumulative Re-CPU Cost: 7.99827e+09

Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 712543

Estimated Bufferpool Buffers: 1

Remote communication cost:26483.8

Arguments:

BLDLEVEL: (Build level)

DB2 v8.1.1.64 : s040509

ENVVAR : (Environment Variable)

DB2_EXTENDED_OPTIMIZATION = ON

STMTHEAP: (Statement heap size)

8192

Input Streams:

27) From Operator #2

Estimated number of rows: 0.41902
Number of columns: 8
Subquery predicate ID: Not Applicable

Column Names:

+Q12.S_ACCTBAL(D)+Q12.N_NAME(A)+Q12.S_NAME(A)
+Q12.P_PARTKEY(A)+Q12.S_COMMENT+Q12.S_PHONE
+Q12.S_ADDRESS+Q12.P_MFGR

2) NLJOIN: (Nested Loop Join)
Cumulative Total Cost: 712543
Cumulative CPU Cost: 7.99833e+09
Cumulative I/O Cost: 31800.2
Cumulative Re-Total Cost: 712268
Cumulative Re-CPU Cost: 7.99827e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 712543
Estimated Bufferpool Buffers: 1
Remote communication cost:26483.8

Arguments:

EARLYOUT: (Early Out flag)
NONE
FETCHMAX: (Override for FETCH MAXPAGES)
IGNORE
ISCANMAX: (Override for ISCAN MAXPAGES)
IGNORE

Input Streams:

24) From Operator #3

Estimated number of rows: 0.41902
Number of columns: 8
Subquery predicate ID: Not Applicable

Column Names:

+Q10.S_ACCTBAL(D)+Q8.N_NAME(A)+Q10.S_NAME(A)
+Q11.P_PARTKEY(A)+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q11.P_MFGR

26) From Operator #31

Estimated number of rows: 1
Number of columns: 0

Subquery predicate ID: Not Applicable

Output Streams:

27) To Operator #1

Estimated number of rows: 0.41902

Number of columns: 8

Subquery predicate ID: Not Applicable

Column Names:

+Q12.S_ACCTBAL(D)+Q12.N_NAME(A)+Q12.S_NAME(A)

+Q12.P_PARTKEY(A)+Q12.S_COMMENT+Q12.S_PHONE

+Q12.S_ADDRESS+Q12.P_MFGR

3) TBSCAN: (Table Scan)

Cumulative Total Cost: 712543

Cumulative CPU Cost: 7.99832e+09

Cumulative I/O Cost: 31800.2

Cumulative Re-Total Cost: 712268

Cumulative Re-CPU Cost: 7.99827e+09

Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 712543

Estimated Bufferpool Buffers: 0

Remote communication cost:26477.6

Arguments:

JN INPUT: (Join input leg)

OUTER

MAXPAGES: (Maximum pages for prefetch)

ALL

PREFETCH: (Type of Prefetch)

NONE

SCANDIR : (Scan Direction)

FORWARD

Input Streams:

23) From Operator #4

Estimated number of rows: 0.41902

Number of columns: 8

Subquery predicate ID: Not Applicable

Column Names:

```

-----
+Q10.S_ACCTBAL(D)+Q8.N_NAME(A)+Q10.S_NAME(A)
+Q11.P_PARTKEY(A)+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q11.P_MFGR

```

Output Streams:

----- 24) To Operator #2

```

Estimated number of rows: 0.41902
Number of columns: 8
Subquery predicate ID: Not Applicable

```

Column Names:

```

-----
+Q10.S_ACCTBAL(D)+Q8.N_NAME(A)+Q10.S_NAME(A)
+Q11.P_PARTKEY(A)+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q11.P_MFGR

```

4) SORT : (Sort)

```

Cumulative Total Cost: 712543
Cumulative CPU Cost: 7.99831e+09
Cumulative I/O Cost: 31800.2
Cumulative Re-Total Cost: 712268
Cumulative Re-CPU Cost: 7.99827e+09
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 712543
Estimated Bufferpool Buffers: 31791.2
Remote communication cost:26477.6

```

Arguments:

```

-----
DUPLWARN: (Duplicates Warning flag)
FALSE
NUMROWS : (Estimated number of rows)
1
ROWWIDTH: (Estimated width of rows)
204
SORTKEY : (Sort Key column)
1: Q10.S_ACCTBAL(D)
SORTKEY : (Sort Key column)
2: Q8.N_NAME(A)
SORTKEY : (Sort Key column)
3: Q10.S_NAME(A)
SORTKEY : (Sort Key column)
4: Q11.P_PARTKEY(A)
TEMPSIZE: (Temporary Table Page Size)

```

4096
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

22) From Operator #5

Estimated number of rows: 0.41902
Number of columns: 11
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0+Q8.N_NAME+Q9.PS_SUPPLYCOST
+Q10.S_COMMENT+Q10.S_PHONE+Q10.S_ADDRESS
+Q10.S_NAME+Q10.S_ACCTBAL+Q10.S_NATIONKEY
+Q11.P_MFGR+Q11.P_PARTKEY

Output Streams:

23) To Operator #3

Estimated number of rows: 0.41902
Number of columns: 8
Subquery predicate ID: Not Applicable

Column Names:

+Q10.S_ACCTBAL(D)+Q8.N_NAME(A)+Q10.S_NAME(A)
+Q11.P_PARTKEY(A)+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q11.P_MFGR

5) NLJOIN: (Nested Loop Join)
Cumulative Total Cost: 712543
Cumulative CPU Cost: 7.99831e+09
Cumulative I/O Cost: 31800.2
Cumulative Re-Total Cost: 712268
Cumulative Re-CPU Cost: 7.99827e+09
Cumulative Re-I/O Cost: 31789.2
Cumulative First Row Cost: 712543
Estimated Bufferpool Buffers: 31791.2
Remote communication cost:26477.6

Arguments:

EARLYOUT: (Early Out flag)

NONE
FETCHMAX: (Override for FETCH MAXPAGES)
IGNORE
ISCANMAX: (Override for ISCAN MAXPAGES)
IGNORE

Predicates:

4) Predicate used in Join
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 0.04

Predicate Text:

(Q10.S_NATIONKEY = Q8.N_NATIONKEY)

Input Streams:

19) From Operator #6

Estimated number of rows: 0.41902
Number of columns: 10
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0+Q9.PS_SUPPLYCOST+Q10.S_COMMENT
+Q10.S_PHONE+Q10.S_ADDRESS+Q10.S_NAME
+Q10.S_ACCTBAL+Q10.S_NATIONKEY+Q11.P_MFGR
+Q11.P_PARTKEY

21) From Operator #29

Estimated number of rows: 1
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q8.N_NAME

Output Streams:

22) To Operator #4

Estimated number of rows: 0.41902

Number of columns: 11
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0+Q8.N_NAME+Q9.PS_SUPPLYCOST
+Q10.S_COMMENT+Q10.S_PHONE+Q10.S_ADDRESS
+Q10.S_NAME+Q10.S_ACCTBAL+Q10.S_NATIONKEY
+Q11.P_MFGR+Q11.P_PARTKEY

6) NLJOIN: (Nested Loop Join)
Cumulative Total Cost: 712543
Cumulative CPU Cost: 7.99829e+09
Cumulative I/O Cost: 31800.2
Cumulative Re-Total Cost: 712268
Cumulative Re-CPU Cost: 7.99826e+09
Cumulative Re-I/O Cost: 31789.2
Cumulative First Row Cost: 712543
Estimated Bufferpool Buffers: 31790.2
Remote communication cost:26471.4

Arguments:

EARLYOUT: (Early Out flag)
NONE
FETCHMAX: (Override for FETCH MAXPAGES)
IGNORE
ISCANMAX: (Override for ISCAN MAXPAGES)
IGNORE
JN INPUT: (Join input leg)
OUTER

Predicates:

2) Predicate used in Join
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 1.16257e-05

Predicate Text:

(Q9.PS_SUPPLYCOST = Q6.\$C0)

Input Streams:

4) From Operator #7

Estimated number of rows: 36042.4
Number of columns: 9
Subquery predicate ID: Not Applicable

Column Names:

+Q9.PS_SUPPLYCOST+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q10.S_NAME+Q10.S_ACCTBAL
+Q10.S_NATIONKEY+Q11.P_MFGR+Q11.P_PARTKEY

18) From Operator #13

Estimated number of rows: 1.16257e-05
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0

Output Streams:

19) To Operator #5

Estimated number of rows: 0.41902
Number of columns: 10
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0+Q9.PS_SUPPLYCOST+Q10.S_COMMENT
+Q10.S_PHONE+Q10.S_ADDRESS+Q10.S_NAME
+Q10.S_ACCTBAL+Q10.S_NATIONKEY+Q11.P_MFGR
+Q11.P_PARTKEY

7) SHIP : (Ship)

Cumulative Total Cost: 708711
Cumulative CPU Cost: 5.90815e+08
Cumulative I/O Cost: 31789.2
Cumulative Re-Total Cost: 708711
Cumulative Re-CPU Cost: 5.90815e+08
Cumulative Re-I/O Cost: 31789.2
Cumulative First Row Cost: 708711
Estimated Bufferpool Buffers: 31790.2
Remote communication cost:26449.2

Arguments:

```

-----
CSERQY : (Remote common subexpression)
      FALSE
DSTSEVER: (Destination (ship to) server)
      - (NULL).
JN INPUT: (Join input leg)
      OUTER
RMTQTX : (Remote statement)
      SELECT A0."P_PARTKEY", A0."P_MFGR", A2."S_NATIONKEY", A2."S_ACCTBAL",
A2."S_NAME", A2."S_ADDRESS", A2."S_PHONE", A2."S_COMMENT", A1."PS_SUPPLYCOST"
FROM "TPCD"."PART" A0, "TPCD"."PARTSUPP" A1, "TPCD"."SUPPLIER" A2 WHERE
(A0."P_SIZE" = 15) AND (A0."P_TYPE" LIKE '%BRASS') AND (A0."P_PARTKEY" =
A1."PS_PARTKEY") AND (A2."S_SUPPKEY" = A1."PS_SUPPKEY") FOR READ ONLY
SRCSEVER: (Source (ship from) server)
      DB2SERV
STREAM : (Remote stream)
      FALSE

```

Input Streams:

----- 1) From Object DB2.PART

```

Estimated number of rows: 2e+06
Number of columns: 5
Subquery predicate ID: Not Applicable

```

Column Names:

```

-----
+Q11.$RID$+Q11.P_MFGR+Q11.P_TYPE+Q11.P_SIZE
+Q11.P_PARTKEY

```

2) From Object DB2.PARTSUPP

```

Estimated number of rows: 8e+06
Number of columns: 4
Subquery predicate ID: Not Applicable

```

Column Names:

```

-----
+Q9.$RID$+Q9.PS_SUPPLYCOST+Q9.PS_SUPPKEY
+Q9.PS_PARTKEY

```

3) From Object DB2.SUPPLIER

```

Estimated number of rows: 100000
Number of columns: 8
Subquery predicate ID: Not Applicable

```

Column Names:

```

-----
+Q10.$RID$+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q10.S_NAME+Q10.S_ACCTBAL
+Q10.S_NATIONKEY+Q10.S_SUPPKEY

```

Output Streams:

4) To Operator #6

```

Estimated number of rows: 36042.4
Number of columns: 9
Subquery predicate ID: Not Applicable

```

Column Names:

```

-----
+Q9.PS_SUPPLYCOST+Q10.S_COMMENT+Q10.S_PHONE
+Q10.S_ADDRESS+Q10.S_NAME+Q10.S_ACCTBAL
+Q10.S_NATIONKEY+Q11.P_MFGR+Q11.P_PARTKEY

```

13) FILTER: (Filter)

```

Cumulative Total Cost: 275.117
Cumulative CPU Cost: 242920
Cumulative I/O Cost: 11
Cumulative Re-Total Cost: 275.117
Cumulative Re-CPU Cost: 242920
Cumulative Re-I/O Cost: 11
Cumulative First Row Cost: 275.117
Estimated Bufferpool Buffers: 0
Remote communication cost:22.2188

```

Arguments:

```

-----
JN INPUT: (Join input leg)
INNER

```

Predicates:

2) Residual Predicate

```

Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 1.16257e-05

```

Predicate Text:

```

-----
(Q9.PS_SUPPLYCOST = Q6.$C0)

```

Input Streams:

17) From Operator #14

Estimated number of rows: 1

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0

Output Streams:

18) To Operator #6

Estimated number of rows: 1.16257e-05

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0

14) GRPBY : (Group By)

Cumulative Total Cost: 275.116

Cumulative CPU Cost: 241585

Cumulative I/O Cost: 11

Cumulative Re-Total Cost: 275.116

Cumulative Re-CPU Cost: 241585

Cumulative Re-I/O Cost: 11

Cumulative First Row Cost: 275.116

Estimated Bufferpool Buffers: 0

Remote communication cost:22.2188

Arguments:

AGGMODE : (Aggregation Mode)

COMPLETE

GROUPBYC: (Group By columns)

FALSE

GROUPBYN: (Number of Group By columns)

0

ONEFETCH: (One Fetch flag)

FALSE

Input Streams:

16) From Operator #15

Estimated number of rows: 0.8
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q5.PS_SUPPLYCOST

Output Streams:

17) To Operator #13

Estimated number of rows: 1
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q6.\$C0

15) MSJOIN: (Merge Scan Join)

Cumulative Total Cost: 275.116
Cumulative CPU Cost: 241135
Cumulative I/O Cost: 11
Cumulative Re-Total Cost: 275.116
Cumulative Re-CPU Cost: 241135
Cumulative Re-I/O Cost: 11
Cumulative First Row Cost: 275.116
Estimated Bufferpool Buffers: 0
Remote communication cost:22.2188

Arguments:

EARLYOUT: (Early Out flag)
NONE
INNERCOL: (Inner Order Columns)
1: Q2.N_NATIONKEY(A)
OUTERCOL: (Outer Order columns)
1: Q3.S_NATIONKEY(A)
TEMPSIZE: (Temporary Table Page Size)
4096

Predicates:

11) Predicate used in Join
Relational Operator: Equal (=)
Subquery Input Required: No
Filter Factor: 0.04

Predicate Text:

(Q3.S_NATIONKEY = Q2.N_NATIONKEY)

Input Streams:

9) From Operator #16

Estimated number of rows: 4
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q3.S_NATIONKEY(A)+Q4.PS_SUPPLYCOST

15) From Operator #22

Estimated number of rows: 0.2
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

Output Streams:

16) To Operator #14

Estimated number of rows: 0.8
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q5.PS_SUPPLYCOST

16) TBSCAN: (Table Scan)
Cumulative Total Cost: 275.092
Cumulative CPU Cost: 191030

Cumulative I/O Cost: 11
Cumulative Re-Total Cost: 25.0504
Cumulative Re-CPU Cost: 104891
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 275.09
Estimated Bufferpool Buffers: 0
Remote communication cost:10.8594

Arguments:

JN INPUT: (Join input leg)
 OUTER
MAXPAGES: (Maximum pages for prefetch)
 ALL
PREFETCH: (Type of Prefetch)
 NONE
SCANDIR : (Scan Direction)
 FORWARD

Input Streams:

8) From Operator #17

 Estimated number of rows: 4
 Number of columns: 2
 Subquery predicate ID: Not Applicable

 Column Names:

 +Q3.S_NATIONKEY(A)+Q4.PS_SUPPLYCOST

Output Streams:

9) To Operator #15

 Estimated number of rows: 4
 Number of columns: 2
 Subquery predicate ID: Not Applicable

 Column Names:

 +Q3.S_NATIONKEY(A)+Q4.PS_SUPPLYCOST

17) SORT : (Sort)
 Cumulative Total Cost: 275.089
 Cumulative CPU Cost: 185333
 Cumulative I/O Cost: 11

Cumulative Re-Total Cost: 25.0476
Cumulative Re-CPU Cost: 99194
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 275.089
Estimated Bufferpool Buffers: 50012
Remote communication cost:10.8594

Arguments:

DUPLWARN: (Duplicates Warning flag)
FALSE
NUMROWS : (Estimated number of rows)
4
ROWWIDTH: (Estimated width of rows)
16
SORTKEY : (Sort Key column)
1: Q3.S_NATIONKEY(A)
TEMPSIZE: (Temporary Table Page Size)
4096
UNIQUE : (Uniqueness required flag)
FALSE

Input Streams:

7) From Operator #18

Estimated number of rows: 4
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q3.S_NATIONKEY+Q4.PS_SUPPLYCOST

Output Streams:

8) To Operator #16

Estimated number of rows: 4
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q3.S_NATIONKEY(A)+Q4.PS_SUPPLYCOST

18) SHIP : (Ship)

Cumulative Total Cost: 275.086
Cumulative CPU Cost: 180037
Cumulative I/O Cost: 11
Cumulative Re-Total Cost: 25.0476
Cumulative Re-CPU Cost: 99194
Cumulative Re-I/O Cost: 1
Cumulative First Row Cost: 125.044
Estimated Bufferpool Buffers: 50012
Remote communication cost:10.8594

Arguments:

CSERQY : (Remote common subexpression)
FALSE
DSTSEVER: (Destination (ship to) server)
- (NULL).
RMTQTX : (Remote statement)
SELECT A0."PS_SUPPLYCOST", A1."S_NATIONKEY" FROM "TPCD"."PARTSUPP"
A0, "TPCD"."SUPPLIER" A1 WHERE (:HO = A0."PS_PARTKEY") AND (A1."S_SUPPKEY" =
A0."PS_SUPPKEY") FOR READ ONLY
SRCSEVER: (Source (ship from) server)
DB2SERV
STREAM : (Remote stream)
FALSE

Input Streams:

5) From Object DB2.PARTSUPP

Estimated number of rows: 8e+06
Number of columns: 4
Subquery predicate ID: Not Applicable

Column Names:

+Q4.\$RID\$+Q4.PS_SUPPLYCOST+Q4.PS_SUPPKEY
+Q4.PS_PARTKEY

6) From Object DB2.SUPPLIER

Estimated number of rows: 100000
Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:

+Q3.\$RID\$+Q3.S_NATIONKEY+Q3.S_SUPPKEY

Output Streams:

7) To Operator #17

Estimated number of rows: 4

Number of columns: 2

Subquery predicate ID: Not Applicable

Column Names:

+Q3.S_NATIONKEY+Q4.PS_SUPPLYCOST

22) FILTER: (Filter)

Cumulative Total Cost: 0.022021

Cumulative CPU Cost: 45856.4

Cumulative I/O Cost: 0

Cumulative Re-Total Cost: 0.00924657

Cumulative Re-CPU Cost: 19255

Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 0.0196007

Estimated Bufferpool Buffers: 0

Remote communication cost:11.3594

Arguments:

JN INPUT: (Join input leg)

INNER

Predicates:

11) Residual Predicate

Relational Operator: Equal (=)

Subquery Input Required: No

Filter Factor: 0.04

Predicate Text:

(Q3.S_NATIONKEY = Q2.N_NATIONKEY)

Input Streams:

14) From Operator #23

Estimated number of rows: 5

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

Output Streams:

15) To Operator #15

Estimated number of rows: 0.2
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

23) TBSCAN: (Table Scan)

Cumulative Total Cost: 0.022021
Cumulative CPU Cost: 45856.4
Cumulative I/O Cost: 0
Cumulative Re-Total Cost: 0.00924657
Cumulative Re-CPU Cost: 19255
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 0.0196007
Estimated Bufferpool Buffers: 0
Remote communication cost:11.3594

Arguments:

MAXPAGES: (Maximum pages for prefetch)
ALL
PREFETCH: (Type of Prefetch)
NONE
SCANDIR : (Scan Direction)
FORWARD

Input Streams:

13) From Operator #24

Estimated number of rows: 5
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

Output Streams:

14) To Operator #22

Estimated number of rows: 5

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

24) SORT : (Sort)

Cumulative Total Cost: 0.0189106

Cumulative CPU Cost: 39379.4

Cumulative I/O Cost: 0

Cumulative Re-Total Cost: 0.00613621

Cumulative Re-CPU Cost: 12778

Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 0.0189106

Estimated Bufferpool Buffers: 2

Remote communication cost:11.3594

Arguments:

DUPLWARN: (Duplicates Warning flag)

FALSE

NUMROWS : (Estimated number of rows)

5

ROWWIDTH: (Estimated width of rows)

8

SORTKEY : (Sort Key column)

1: Q2.N_NATIONKEY(A)

TEMPSIZE: (Temporary Table Page Size)

4096

UNIQUE : (Uniqueness required flag)

FALSE

Input Streams:

12) From Operator #25

Estimated number of rows: 5

Number of columns: 1

Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY

Output Streams:

13) To Operator #23

Estimated number of rows: 5
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY(A)

25) SHIP : (Ship)

Cumulative Total Cost: 0.0160508
Cumulative CPU Cost: 33424
Cumulative I/O Cost: 0
Cumulative Re-Total Cost: 0.00613621
Cumulative Re-CPU Cost: 12778
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 0.00919759
Estimated Bufferpool Buffers: 2
Remote communication cost:11.3594

Arguments:

CSERQY : (Remote common subexpression)
FALSE

DSTSEVER: (Destination (ship to) server)
- (NULL).

RMTQTX : (Remote statement)

SELECT A1."N_NATIONKEY" FROM "IITEST"."REGION" A0, "IITEST"."NATION"
A1 WHERE (A0."R_NAME" = 'EUROPE ') AND (A1."N_REGIONKEY" =
A0."R_REGIONKEY")

SRCSEVER: (Source (ship from) server)

ORASERV

STREAM : (Remote stream)

FALSE

Input Streams:

10) From Object ORA.REGION

Estimated number of rows: 5

Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:

+Q1.\$RID\$+Q1.R_NAME+Q1.R_REGIONKEY

11) From Object ORA.NATION

Estimated number of rows: 25
Number of columns: 3
Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$+Q2.N_REGIONKEY+Q2.N_NATIONKEY

Output Streams:

12) To Operator #24

Estimated number of rows: 5
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:

+Q2.N_NATIONKEY

29) SHIP : (Ship)

Cumulative Total Cost: 0.00605841
Cumulative CPU Cost: 12616
Cumulative I/O Cost: 0
Cumulative Re-Total Cost: 0.00399012
Cumulative Re-CPU Cost: 8309
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 0.00498321
Estimated Bufferpool Buffers: 1
Remote communication cost:9.35938

Arguments:

CSERQY : (Remote common subexpression)
FALSE
DSTSEVER: (Destination (ship to) server)
- (NULL).
JN INPUT: (Join input leg)


```

      INNER
      RMTQTX : (Remote statement)
      SELECT AO."N_NAME" FROM "IITEST"."NATION" AO WHERE (:HO =
AO."N_NATIONKEY")
      SRCSEVER: (Source (ship from) server)
      ORASERV
      STREAM  : (Remote stream)
      FALSE

```

Input Streams:

20) From Object ORA.NATION

```

      Estimated number of rows: 25
      Number of columns: 3
      Subquery predicate ID: Not Applicable

```

Column Names:

+Q8.\$RID\$+Q8.N_NAME+Q8.N_NATIONKEY

Output Streams:

21) To Operator #5

```

      Estimated number of rows: 1
      Number of columns: 1
      Subquery predicate ID: Not Applicable

```

Column Names:

+Q8.N_NAME

31) SHIP : (Ship)

```

      Cumulative Total Cost: 0.00418893
      Cumulative CPU Cost: 8723
      Cumulative I/O Cost: 0
      Cumulative Re-Total Cost: 0.00214609
      Cumulative Re-CPU Cost: 4469
      Cumulative Re-I/O Cost: 0
      Cumulative First Row Cost: 0.00313918
      Estimated Bufferpool Buffers: 1
      Remote communication cost:9.35938

```

Arguments:

CSERQY : (Remote common subexpression)

```

        FALSE
DSTSEVER: (Destination (ship to) server)
- (NULL).
JN INPUT: (Join input leg)
INNER
RMTQTX : (Remote statement)
        SELECT '1' FROM "IITEST"."REGION" AO WHERE (AO."R_NAME" = 'EUROPE
')
SRCSEVER: (Source (ship from) server)
ORASERV
STREAM  : (Remote stream)
        FALSE

```

Input Streams:

25) From Object ORA.REGION

```

Estimated number of rows: 5
Number of columns: 2
Subquery predicate ID: Not Applicable

```

Column Names:

+Q7.\$RID\$+Q7.R_NAME

Output Streams:

26) To Operator #2

```

Estimated number of rows: 1
Number of columns: 0
Subquery predicate ID: Not Applicable

```

Objects Used in Access Plan:

Schema: DB2

Name: PART

Type: Nickname

Time of creation: 2004-06-10-08.38.47.722759

Last statistics update:

Number of columns: 9

Number of rows: 2000000

Width of rows: 70

Number of buffer pool pages: 76238

Distinct row values: No

Tablespace name:

Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: DB2

Name: PARTSUPP

Type: Nickname

Time of creation: 2004-06-10-08.38.47.872025
Last statistics update:
Number of columns: 5
Number of rows: 8000000
Width of rows: 28
Number of buffer pool pages: 319290
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: DB2

Name: SUPPLIER

Type: Nickname

Time of creation: 2004-06-10-08.38.47.795922
Last statistics update:
Number of columns: 7
Number of rows: 100000
Width of rows: 157
Number of buffer pool pages: 4093
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: ORA

Name: NATION

Type: Nickname

Time of creation: 2004-06-10-08.38.38.710626
Last statistics update: 2004-06-10-19.56.42.522824
Number of columns: 4
Number of rows: 25
Width of rows: 57
Number of buffer pool pages: 2

Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Schema: ORA
Name: REGION
Type: Nickname
Time of creation: 2004-06-10-08.38.38.810764
Last statistics update: 2004-06-10-19.56.42.820988
Number of columns: 3
Number of rows: 5
Width of rows: 53
Number of buffer pool pages: 2
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Access plan description

The access plan graph for our SQL query has been highlighted in Example B-13 on page 586 under “Access Plan”.

The key difference between this access plan graph and the one with `DB2_MAXIMAL_PUSHDOWN = 'N'` in Example B-12 on page 554 is that a 3-way join of `PART`, `PARTSUPP`, and `SUPPLIER` has now been pushed down to `DB2` (see `SHIP` operator 7), thereby eliminating a need for a nested loop join. However, the estimated total cost is now 712543 timerons, which is higher than before.

Analysis

Important: Bearing in mind that the EXPLAIN output information is `DB2` optimizer estimates only and does not necessarily reflect actual runtime metrics, you should be cautious in drawing conclusions about relative costs associated with each operator, as well as relying on the number of estimated rows retrieved or returned by the various operators. EXPLAIN output information is best used in conjunction with runtime metrics gathered through monitoring.

Even though additional pushdown has occurred for our SQL query with DB2_MAXIMAL_PUSHDOWN = 'Y', the optimizer has estimated the costs to be higher.

We recommend that you choose the default DB2_MAXIMAL_PUSHDOWN = 'N' setting for all queries, and only consider changing this default if performance is unacceptable and a change in the setting delivers superior performance.

Attention: The DB2_MAXIMAL_PUSHDOWN can be set using the SET SERVER OPTION statement so that its impact can be limited to a particular query. Changing this default setting in the server definition can negatively impact other queries that are better served with the default setting.

Note: There is no indication of the server option (DB2_MAXIMAL_PUSHDOWN set to 'Y' or 'N') in the **db2exfmt** output.

The server options are stored in the DB2 system catalog table SYSIBM.SYSSERVEROPTIONS or view SYSCAT.SERVEROPTIONS.

SQL INSERT/UPDATE/DELETE

In this scenario, we merely want to show that a SHIP operator may or may not appear in the **db2exfmt** output for inserts/updates/deletes, depending upon the syntax used.

Example B-14, Example B-15 on page 624, and Example B-16 on page 627 show the complete **db2exfmt** output for the insert, update, and delete DML statements that reference remote nicknames. The SQL statement issued in each of these examples has been highlighted under "Original statement" in the **db2exfmt** output.

Example: B-14 db2exfmt output for SQL INSERT

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

***** EXPLAIN INSTANCE *****

DB2_VERSION: 08.02.0
SOURCE_NAME: SQLC2E05
SOURCE_SCHEMA: NULLID

SOURCE_VERSION:
EXPLAIN_TIME: 2004-06-28-14.53.19.283679
EXPLAIN_REQUESTER: DB2I32

Database Context:

Parallelism: None
CPU Speed: 4.802167e-07
Comm Speed: 100
Buffer Pool size: 75000
Sort Heap size: 20000
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 1020

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 203 -----

QUERYNO: 1
QUERYTAG:
Statement Type: Insert
Updatable: Not Applicable
Deletable: Not Applicable
Query Degree: 1

Original Statement:

**insert into ora.nation
select *
from tpcd.nation**

Optimized Statement:

INSERT INTO ORA.NATION AS Q3
SELECT Q1.N_COMMENT, Q1.N_NAME, DECIMAL(Q1.N_NATIONKEY),
DECIMAL(Q1.N_REGIONKEY)
FROM TPCD.NATION AS Q1

Access Plan:

Total Cost: 50.0434
Query Degree:1

```

      Rows
      RETURN
      ( 1)
      Cost
      I/O
      |
      25
      SHIP
      ( 3)
      25.0372
      1
      |
      25
      FETCH
      ( 4)
      25.0316
      1
      /-----\
      25          25
      IXSCAN      TABLE: TPCD
      ( 5)         NATION
      0.00330581
      0
      |
      25
      INDEX: TPCD
      N_REGKEYNATKEYNA
```

1) RETURN: (Return Result)
Cumulative Total Cost: 50.0434
Cumulative CPU Cost: 90430
Cumulative I/O Cost: 2
Cumulative Re-Total Cost: 25.0354
Cumulative Re-CPU Cost: 73725
Cumulative Re-I/O Cost: 1
Cumulative First Row Cost: 50.0434
Estimated Bufferpool Buffers: 3
Remote communication cost:6.144e+06

Arguments:

BLDLEVEL: (Build level)

```

DB2 v8.1.1.64 : s040509
ENVVAR : (Environment Variable)
DB2_EXTENDED_OPTIMIZATION = ON
RMTQTX : (Remote statement)
INSERT INTO "IITEST"."NATION" ( "N_REGIONKEY", "N_NATIONKEY",
"N_NAME", "N_COMMENT") VALUES ( :H0 , :H1 , :H2 , :H3 )
RMTSEVER: (Remote server)
ORASERV
STMTHEAP: (Statement heap size)
8192

```

Input Streams:

5) From Operator #3

```

Estimated number of rows: 25
Number of columns: 0
Subquery predicate ID: Not Applicable

```

```

3) SHIP : (Ship)
Cumulative Total Cost: 25.0372
Cumulative CPU Cost: 77430
Cumulative I/O Cost: 1
Cumulative Re-Total Cost: 0.0291612
Cumulative Re-CPU Cost: 60725
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 25.0094
Estimated Bufferpool Buffers: 2

```

Arguments:

```

CSERQY : (Remote common subexpression)
FALSE
DSTSEVER: (Destination (ship to) server)
ORASERV
SRCSEVER: (Source (ship from) server)
- (NULL).
STREAM : (Remote stream)
FALSE

```

Input Streams:

4) From Operator #4

```

Estimated number of rows: 25
Number of columns: 4
Subquery predicate ID: Not Applicable

```


Column Names:

+Q2.\$C3+Q2.\$C2+Q2.N_COMMENT+Q2.N_NAME

Output Streams:

5) To Operator #1

Estimated number of rows: 25

Number of columns: 0

Subquery predicate ID: Not Applicable

4) FETCH : (Fetch)

Cumulative Total Cost: 25.0316

Cumulative CPU Cost: 65860

Cumulative I/O Cost: 1

Cumulative Re-Total Cost: 0.0236051

Cumulative Re-CPU Cost: 49155

Cumulative Re-I/O Cost: 0

Cumulative First Row Cost: 25.0092

Estimated Bufferpool Buffers: 2

Arguments:

MAXPAGES: (Maximum pages for prefetch)

ALL

MAXPAGES: (Maximum pages for prefetch)

ALL

PREFETCH: (Type of Prefetch)

NONE

ROWLOCK : (Row Lock intent)

NONE

TABLOCK : (Table Lock intent)

SHARE

TBISOLVL: (Table access Isolation Level)

CURSOR STABILITY

Input Streams:

2) From Operator #5

Estimated number of rows: 25

Number of columns: 3

Subquery predicate ID: Not Applicable

Column Names:

```
+Q1.N_REGIONKEY(A)+Q1.N_NATIONKEY(A)
+Q1.N_NAME(A)
```

3) From Object TPCD.NATION

```
Estimated number of rows: 25
Number of columns: 1
Subquery predicate ID: Not Applicable
```

```
Column Names:
-----
+Q1.N_COMMENT
```

```
Output Streams:
-----
```

4) To Operator #3

```
Estimated number of rows: 25
Number of columns: 4
Subquery predicate ID: Not Applicable
```

```
Column Names:
-----
+Q2.$C3+Q2.$C2+Q2.N_COMMENT+Q2.N_NAME
```

5) IXSCAN: (Index Scan)

```
Cumulative Total Cost: 0.00330581
Cumulative CPU Cost: 6884
Cumulative I/O Cost: 0
Cumulative Re-Total Cost: 0.00152661
Cumulative Re-CPU Cost: 3179
Cumulative Re-I/O Cost: 0
Cumulative First Row Cost: 0.0025197
Estimated Bufferpool Buffers: 1
```

```
Arguments:
-----
```

```
MAXPAGES: (Maximum pages for prefetch)
ALL
PREFETCH: (Type of Prefetch)
NONE
ROWLOCK : (Row Lock intent)
NONE
SCANDIR : (Scan Direction)
FORWARD
TABLOCK : (Table Lock intent)
SHARE
```

Input Streams:

1) From Object TPCD.N_REGKEYNATKEYNAM

Estimated number of rows: 25

Number of columns: 4

Subquery predicate ID: Not Applicable

Column Names:

+Q1.N_REGIONKEY(A)+Q1.N_NATIONKEY(A)

+Q1.N_NAME(A)+Q1.\$RID\$

Output Streams:

2) To Operator #4

Estimated number of rows: 25

Number of columns: 3

Subquery predicate ID: Not Applicable

Column Names:

+Q1.N_REGIONKEY(A)+Q1.N_NATIONKEY(A)

+Q1.N_NAME(A)

Objects Used in Access Plan:

Schema: ORA

Name: NATION

Type: Nickname (reference only)

Schema: TPCD

Name: N_REGKEYNATKEYNAM

Type: Index

Time of creation: 2004-06-09-04.46.41.746356

Last statistics update: 2004-06-13-00.38.53.238586

Number of columns: 3

Number of rows: 25

Width of rows: -1

Number of buffer pool pages: 2

Distinct row values: Yes

Tablespace name: INDEX_TS

Tablespace overhead: 24.100000

Tablespace transfer rate: 0.900000

Source for statistics: Single Node
 Prefetch page count: 96
 Container extent page count: 32
 Index clustering statistic: 100.000000
 Index leaf pages: 1
 Index tree levels: 1
 Index full key cardinality: 25
 Index first key cardinality: 5
 Index first 2 keys cardinality: 25
 Index first 3 keys cardinality: 25
 Index first 4 keys cardinality: -1
 Index sequential pages: 0
 Index page density: 0
 Index avg sequential pages: 0
 Index avg gap between sequences: 0
 Index avg random pages: 1
 Fetch avg sequential pages: -1
 Fetch avg gap between sequences: -1
 Fetch avg random pages: -1
 Index RID count: 25
 Index deleted RID count: 0
 Index empty leaf pages: 0
 Base Table Schema: TPCD
 Base Table Name: NATION
 Columns in index:
 N_REGIONKEY
 N_NATIONKEY
 N_NAME

Schema: TPCD
 Name: NATION
 Type: Table
 Time of creation: 2004-06-09-04.46.40.636983
 Last statistics update: 2004-06-13-00.38.53.238586
 Number of columns: 4
 Number of rows: 25
 Width of rows: 116
 Number of buffer pool pages: 2
 Distinct row values: No
 Tablespace name: DATA_TS
 Tablespace overhead: 24.100000
 Tablespace transfer rate: 0.900000
 Source for statistics: Single Node
 Prefetch page count: 96
 Container extent page count: 32
 Table overflow record count: 0
 Table Active Blocks: -1

Access plan description

Since the format of the insert SQL statement in this example is “insert via subselect” (insert into T1, select from T2), a SHIP operator is present in the **db2exfmt** access plan graph. Had the statement been in the form “insert via values” (insert into T1 values(...)), the SHIP operator would *not* have been present. Also of note, the remote query text (RMTQTXT) for insert, update, and delete statements referencing nicknames show up in the RETURN operator instead of the SHIP operator, as with select DML statements.

The access plan graph for our SQL query has been highlighted in Example B-14 on page 615 under “Access Plan”.

Our SQL query is an insert into the NATION nickname by selecting rows from the local NATION table—an insert with subselect.

Reading this graph bottom up reveals the following:

- ▶ An index scan (IXSXAN operator 5) is performed on the N_REGKEYNATKEYNA index (cardinality of 25 rows according to the DB2 II catalog), which retrieves four columns (N_REGIONKEY, N_NATIONKEY, N_NAME, and the RID) and passes them to the FETCH operator 4, which accesses the N_COMMENT column from the NATION table (same cardinality of 25 as in the index).
- ▶ The FETCH operator 4 estimates that 25 rows will be passed to the SHIP operator 3 with four columns of data.
- ▶ The SHIP operator 3 does not have the RMTQTXT field showing the SQL fragment executed at the remote data source in the Arguments section, as in the case of SQL SELECTs. Instead, the RMTQTXT field is part of the Arguments section of the RETURN operator 1. The RETURN operator 1 is notified of the estimate of 25 rows being inserted.

Attention: The SHIP operator may not appear at all in the **db2exfmt** output when a simple SQL statement such as INSERT INTO T1 VALUES (1,'NAGRAJ', 'ITSO PROJECT LEADER') is used.

- ▶ The total estimated cost of the query is 50.0434 timerons, and there is no parallelism involved (Query Degree: 1 in the Access Plan).

Analysis

Important: Bearing in mind that the EXPLAIN output information is DB2 optimizer estimates only and does not necessarily reflect actual runtime metrics, you should be cautious in drawing conclusions about relative costs associated with each operator, as well as relying on the number of estimated rows retrieved or returned by the various operators. EXPLAIN output information is best used in conjunction with runtime metrics gathered through monitoring.

The key point to be made here is that the RMTQTX field is part of the Arguments section of the RETURN operator, and that a SHIP operator may not appear in the db2exfmt output (this example is not shown here).

Example B-15 and Example B-16 on page 627 show SQL UPDATE and DELETE statements that reference only the remote data source, and therefore do not contain a SHIP operator in the **db2exfmt** output. Note that the RMTQTX field showing the SQL fragment executed at the remote data source is listed in the Arguments section of the RETURN operator 1.

Example: B-15 db2exfmt output for SQL UPDATE

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

***** EXPLAIN INSTANCE *****

DB2_VERSION: 08.02.0
SOURCE_NAME: SQLC2E05
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-06-28-09.37.23.334567
EXPLAIN_REQUESTER: DB2I32

Database Context:

Parallelism: Intra-Partition Parallelism
CPU Speed: 4.841528e-07
Comm Speed: 100
Buffer Pool size: 1000
Sort Heap size: 256
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10

Average Applications: 1
Locks Available: 1020

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors
Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 203 -----

QUERYNO: 1
QUERYTAG:
Statement Type: Searched Update
Udatable: Not Applicable
Deletable: Not Applicable
Query Degree: -1

Original Statement:

**update ora.nation set n_comment = 'updated DC'
where n_nationkey=50**

Optimized Statement:

**UPDATE ORA.NATION AS Q1 SET (Q1.N_COMMENT) =
SELECT 'updated DC'
FROM ORA.NATION AS Q2
WHERE (Q2.N_NATIONKEY = +0000000050.)**

Access Plan:

**Total Cost: 50.0189
Query Degree:0**

**Rows
RETURN
(1)
Cost
I/O
|
25
NICKNM: ORA
NATION**

```

1) RETURN: (Return Result)
Cumulative Total Cost: 50.0189
Cumulative CPU Cost: 39090
Cumulative I/O Cost: 2
Cumulative Re-Total Cost: 25.012
Cumulative Re-CPU Cost: 24785
Cumulative Re-I/O Cost: 1
Cumulative First Row Cost: 50.0189
Estimated Bufferpool Buffers: 3
Remote communication cost:6.144e+06

Arguments:
-----
BLDLEVEL: (Build level)
DB2 v8.1.1.64 : s040509
ENVVAR : (Environment Variable)
DB2_EXTENDED_OPTIMIZATION = ON
RMTQTX : (Remote statement)
UPDATE "IITEST"."NATION" AO SET "N_COMMENT" = 'updated DC' WHERE
(AO."N_NATIONKEY" = 0000000050.)
RMTSEVER: (Remote server)
ORASERV
STMTHEAP: (Statement heap size)
8192

Input Streams:
-----
1) From Object ORA.NATION

Estimated number of rows: 25
Number of columns: 1
Subquery predicate ID: Not Applicable

Column Names:
-----
+Q2.N_COMMENT

Objects Used in Access Plan:
-----

Schema: ORA
Name: NATION
Type: Nickname
Time of creation: 2004-06-10-08.38.38.710626
Last statistics update: 2004-06-10-19.56.42.522824
Number of columns: 4

```


Number of rows: 25
Width of rows: 107
Number of buffer pool pages: 2
Distinct row values: No
Tablespace name:
Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Example: B-16 db2exfmt output for SQL DELETE

DB2 Universal Database Version 8.1, 5622-044 (c) Copyright IBM Corp. 1991, 2002
Licensed Material - Program Property of IBM
IBM DATABASE 2 Explain Table Format Tool

***** EXPLAIN INSTANCE *****

DB2_VERSION: 08.02.0
SOURCE_NAME: SQLC2E05
SOURCE_SCHEMA: NULLID
SOURCE_VERSION:
EXPLAIN_TIME: 2004-06-28-09.34.28.709947
EXPLAIN_REQUESTER: DB2I32

Database Context:

Parallelism: Intra-Partition Parallelism
CPU Speed: 4.841528e-07
Comm Speed: 100
Buffer Pool size: 1000
Sort Heap size: 256
Database Heap size: 1200
Lock List size: 100
Maximum Lock List: 10
Average Applications: 1
Locks Available: 1020

Package Context:

SQL Type: Dynamic
Optimization Level: 5
Blocking: Block All Cursors

Isolation Level: Cursor Stability

----- STATEMENT 1 SECTION 203 -----

QUERYNO: 1
QUERYTAG:
Statement Type: Searched Delete
Updatable: Not Applicable
Deletable: Not Applicable
Query Degree: -1

Original Statement:

delete
from ora.nation
where n_nationkey=50

Optimized Statement:

DELETE
FROM ORA.NATION AS Q1
WHERE \$RID\$ IN
 (SELECT \$RID\$
 FROM ORA.NATION AS Q2
 WHERE (Q2.N_NATIONKEY = +0000000050.))

Access Plan:

Total Cost: 25.0124
Query Degree:0

Rows
RETURN
(1)
Cost
I/O
|
25
NICKNM: ORA
NATION

1) RETURN: (Return Result)
Cumulative Total Cost: 25.0124
Cumulative CPU Cost: 25542

Cumulative I/O Cost: 1
Cumulative Re-Total Cost: 25.0103
Cumulative Re-CPU Cost: 21309
Cumulative Re-I/O Cost: 1
Cumulative First Row Cost: 25.0124
Estimated Bufferpool Buffers: 2
Remote communication cost:6.144e+06

Arguments:

BLDLEVEL: (Build level)
DB2 v8.1.1.64 : s040509
ENVVAR : (Environment Variable)
DB2_EXTENDED_OPTIMIZATION = ON
RMTQTX : (Remote statement)
DELETE FROM "IITEST"."NATION" AO WHERE (AO."N_NATIONKEY" =
0000000050.)
RMTSEVER: (Remote server)
ORASERV
STMTHEAP: (Statement heap size)
8192

Input Streams:

1) From Object ORA.NATION

Estimated number of rows: 25
Number of columns: 2
Subquery predicate ID: Not Applicable

Column Names:

+Q2.\$RID\$+Q2.N_NATIONKEY

Objects Used in Access Plan:

Schema: ORA
Name: NATION
Type: Nickname
Time of creation: 2004-06-10-08.38.38.710626
Last statistics update: 2004-06-10-19.56.42.522824
Number of columns: 4
Number of rows: 25
Width of rows: 34
Number of buffer pool pages: 2
Distinct row values: No
Tablespace name:

Tablespace overhead: 24.100000
Tablespace transfer rate: 0.900000
Source for statistics: Single Node
Prefetch page count: 32
Container extent page count: 32

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 633. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *DB2 UDB ESE V8 non-DPF Performance Guide for High Performance OLTP and BI*, SG24-6432
- ▶ *DB2 UDB's High-Function Business Intelligence in e-business*, SG24-6546
- ▶ *Patterns: Information Aggregation and Data Integration with DB2 Information Integrator*, SG24-7101
- ▶ *WebSphere Portal and DB2 Information Integrator*, SG24-6433
- ▶ *Data Federation with IBM DB2 Information Integrator*, SG24-7052
- ▶ *Fundamentals of Grid Computing*, REDP-3613-00
- ▶ *A Practical Guide to DB2 Data Replication V8*, SG24-6828
- ▶ *Getting Started in Integrating Your Information*, SG24-6892
- ▶ *XML for DB2 Information Integration*, SG24-6994
- ▶ *IBM Life Sciences Solutions: Turning Data into Discovery with DiscoverLink*, SG24-6290
- ▶ *IBM Informix: Integration Through Data Federation*, SG24-7032
- ▶ *Moving Data Across the DB2 Family*, SG24-6905

Other publications

These publications are also relevant as further information sources:

- ▶ *IBM Systems Journal Vol. 41, No. 4, 2002, Information Integration*, G321-01473
- ▶ *IBM DB2 Information Integrator Developer's Guide Version 8.2*, SC18-9174

- ▶ *IBM DB2 Information Integrator Federated Systems Guide Version 8.2*, SC18-7364-01
- ▶ *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows Version 8.2*, GC18-7036-01
- ▶ *IBM DB2 Information Integrator Wrapper Developer's Guide Version 8.2*, SC18-9174
- ▶ *IBM DB2 Information Integrator Migration Guide Version 8.2*, SC18-7360
- ▶ *IBM DB2 Information Integrator Application Developer's Guide Version 8.2*, SC18-7359-01
- ▶ *IBM DB2 UDB ESE non-DPF Performance Guide for High Performance OLTP and BI*, SG24-6432
- ▶ *IBM DB2 Universal Database Administration Guide: Performance Version 8*, SC09-4821-01
- ▶ *IBM DB2 Universal Database Administration Guide: Planning Version 8.2*, SC09-4822-01
- ▶ *IBM DB2 Universal Database Command Reference, Version 8.2*, SC09-4828-01
- ▶ *IBM DB2 Universal Database Message Reference Volume 1 Version 8.2*, GC09-4840-01
- ▶ *IBM DB2 Universal Database Replication Guide and Reference Version 8.2, Release 1*, SC27-1121-02
- ▶ *IBM DB2 Universal Database SQL Reference Volume 1 Version 8.2*, SC09-4844-01
- ▶ *IBM DB2 Universal Database SQL Reference Volume 2 Version 8.2*, SC09-4845-01
- ▶ *IBM DB2 Universal Database System Monitor Guide and Reference Version 8.2*, SC09-4847-01
- ▶ *IBM DB2 Information Integrator Installation Guide for Linux, UNIX, and Windows*, GC18-7036

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ *IBM DB2 Information Integrator Data Source Configuration Guide Version 8.2*, available as softcopy only from the Web site:

<http://www.ibm.com/software/data/integration/solution>

- ▶ *IBM DB2 Information Integrator Release Notes Version 8.2*, available as softcopy only from the Web site:
<http://www.ibm.com/software/data/integration/solution>
- ▶ *Information On Demand* DB2 Magazine article by Holly Hayes and Nelson Mattos, Quarter 3, 2003, Volume 8, Issue 3, available at:
<http://www.db2mag.com/story/showArticle.jhtml?articleID=12803103>
- ▶ *Using the federated database technology of IBM DB2 Information Integrator*, white paper by Anjali Grover, Eileen Lin and Ioana Ursu, available from the Web site:
<http://www.ibm.com/software/data/pubs/papers/#iipapers>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

A

Access 452
Access Plan Graph 85
Access Plan graph 454
alerts 37, 439

B

bandwidth latency 58
best practices guidelines 40
best practices recommendations 40
buffer pool 49, 54, 84, 113
buffer pools 83
built-in functions 24

C

cache 12, 84
cache tables 25, 87–88, 436, 438
Capacity planning 378
capacity planning assumptions 378
capacity planning new applications 427
capacity planning procedure 379
Capacity planning procedure overview
 Step 1 Establish environment 382
 Step 3 Summarize monitored intervals information 410
 Step 5 Generate utilization report 415
 Step 6 Estimate capacity for anticipated growth 424
 Step 2 Capture runtime metrics 386
 Step 4 Identify reporting interval 414
capacity planning procedure overview 381
capture ratio 380, 383, 396, 414–415
CARD 69, 72
cardinality 467
check constraints 89–90, 438
CLUSTERFACTOR 69, 72
CLUSTERRATIO 69, 72
COLCARD 69, 72
collating sequence 78
COLLATING_SEQUENCE 20, 46, 68, 79, 82, 95
COLLATING_SEQUENCE server option 46
collocated federated server 26, 28

COMM_RATE 20, 50, 68, 79, 467, 469
communication buffer 100
compensation 14
computational partition group 435, 468
CPG 29, 62, 64, 160, 468
CPU_RATIO 20, 50, 68, 79, 467, 469
Cube Views 90
CURRENT REFRESH AGE 88

D

Data consolidation or placement 6, 8
data federation 4–5, 11, 16
Data mappings 23
Data placement 60
data placement 25
Data type mappings 23
data type mappings 77
data type mismatch 77, 110
database manager configuration parameter
 intra_parallel 75
 max_querydegree 76
 rqrioblk 99
Database System Monitor 164
db 439
db.fed_nicknames_op_status 439
db.fed_servers_op_status 439
DB2 Enterprise Server Edition 15
DB2 Health Center 125, 439
DB2 hypotheses hierarchy 117
 DB2 resource constraints
 Buffer pool constraints 132
 Cache size constraints 134
 Connection constraints 126
 Locking constraints 130
 Miscellaneous constraints 135
 Sorting constraints 128
DB2 II server options 464
 COMM_RATE 464
 CPU_RATIO 464
 DB2_MAXIMAL_PUSHDOWN 464
 IO_RATIO 464
 LOGIN_TIMEOUT 464
 PACKET_SIZE 464

- PLAN_HINTS 464
- PUSHDOWN 465
- TIMEOUT 465
- VARCHAR_NO_TRAILING_BLANKS 466
- DB2 II V8.1 11, 14
 - components 15
 - data sources supported 15
 - overview 11
- DB2 IICF 10
- DB2 Information Integration
 - overview 6
 - products 9
- DB2 Information Integrator (DB2 II) 10
- DB2 Information Integrator Classic Federation for z/OS (DB2 IICF) 10
- DB2 optimizer 72, 83, 210
- DB2_COMPPARTITIONGROUP 64
- DB2_COMPPARTITIONGROUP registry variable 436
- DB2_FENCED 49, 62, 65, 68, 74, 123, 160, 430, 434
- DB2_MAXIMAL_PUSHDOWN 20, 49, 53, 68, 82, 111, 442, 463, 467, 554, 586
- db2exfmt examples 463
- db2exfmt output 458
 - Access Plan graph 453
 - Access Plan section 452
 - EXPLAIN INSTANCE section 449
 - Objects section 457
 - OPERATOR DETAILS section 455
 - SQL INSERT/UPDATE/DELETE 615
 - SQL STATEMENT section 450
- db2exfmt output example
 - DB2_MAXIMAL_PUSHDOWN = 'Y' 554, 586
 - DPF environment with the FENCED = 'N' 516
 - DPF with FENCED = 'Y' 534
 - INTRA_PARALLEL = YES (intra-partition enabled) 489
 - SQL INSERT/UPDATE/DELETE 615
- db2exfmt output focus areas 463
- db2exfmt overview 448
- dedicated federated server 26, 28
- Default DB2_FENCED wrapper option with DPF 339
- default mapping types 24
- DEGREE 75
- determine new application workload 428
- DFT_DEGREE 48, 68, 74, 160, 433, 468
- dft_degree 75

- DFT_MON_BUFPOOL 162
- DFT_MON_LOCK 162, 164
- DFT_MON_SORT 162
- DFT_MON_STMT 163–164
- DFT_MON_TABLE 163
- DFT_MON_TIMESTAMP 163–164
- DFT_MON_UOW 163
- DFT_MTTB_TYPES 89
- DFT_QUERYOPT 48, 68
- DISABLE QUERY OPTIMIZATION 90
- Distributed access 7, 9
- DPF 62, 160, 378, 463, 468, 516, 534
- dynamic cache 53
- dynamic SQL 52

E

- Efficient SQL queries 109
- EII 7
 - enced mode procedure 431
- enterprise information integration (EII) 7
- estimate capacity for the new application 428
- Estimate the CPU utilization 425
- Estimate the memory requirements 426
- Event Monitor 164
- Exception monitoring 37, 39, 166
- Execution flow of a federated query 50
- EXPLAIN 441
- EXPLAIN facility 442
- EXPLAIN output operators 446
 - BTQ 446
 - DELETE 446
 - DTQ 447
 - EISCAN 447
 - FETCH 447
 - FILTER 447
 - GRPB 447
 - HSJOIN 447
 - INSERT 447
 - IXAND 447
 - IXSCAN 447
 - LMTQ 447
 - LTQ 447
 - MBTQ 447
 - MDTQ 447
 - MSJOIN 447
 - NLJOIN 447
 - RETURN 447
 - RIDSCAN 447

- RPD 448
- SHIP 448
- SORT 448
- TBSCAN 448
- TEMP 448
- TQUEUE 448
- UNION 448
- UNIQUE 448
- UPDATE 448
- EXPLAIN tables 444
 - ADVISE_INDEX 445
 - ADVISE_WORKLOAD 445
 - EXPLAIN_ARGUMENTS 444
 - EXPLAIN_INSTANCE 444
 - EXPLAIN_OBJECTS 444
 - EXPLAIN_OPERATORS 444
 - EXPLAIN_PREDICATE 445
 - EXPLAIN_STATEMENTS 444
 - EXPLAIN_STREAM 445
- Explain tables 444

F

- FEDERATED 16, 18
- federated database 12, 83
- federated query basic flow 42
- federated server 69
- federated server related 152
- federated server side performance 58
- federated system 12, 18, 83
 - configuring a data source 19
- federated test environment 167, 462
- FEDERATED_TOOL 89
- federation 9
- FENCED 463, 516, 534
- fenced mode 430
- fenced mode operation 49
- fenced mode procedure 63
- fenced wrapper 431
- FIRSTKEYCARD 69, 72
- fmp 63
- fmp process 431
- FPAGES 69, 72
- FULLKEYCARD 69, 72
- Function mappings 21
- function mappings 47
- function template 112

G

- get_stats 70
- global catalog 16, 69–71, 73–74, 79, 110, 436
- global catalog views 16
- global optimization 24
- grid computing 3

H

- hash join 49, 83, 158, 467
- hash joins 60
- health indicators 439
- Health Monitor 440
- HIGH2KEY 69, 72
- hypotheses 40
- hypotheses validation 117, 122

I

- IBMDEFAULTBP 84, 96
- incompatible data types on join columns 239
- index definitions 109, 467
- index information 43, 48, 69, 71, 73, 78, 111, 155–156
- index specification 71–74
- index specifications 123
- indexes 69, 73
- information integration 4–6
- informational constraints 89–91, 110, 438
- Input streams 457
- integration
 - Application connectivity 3
 - Build to integrate 3
 - Information integration 3
 - Process integration 3
 - User interaction 3
- inter-partition parallelism 29, 49, 74, 160, 433–434, 464, 468
- INTRA_PARALLEL 48, 68, 74, 91, 123, 160, 433, 463, 468, 516, 554
- intra_parallel 91
- intra-partition parallelism 48, 74–75, 160, 432, 464, 468
- IO_RATIO 20, 50, 68, 79, 467, 469
- IUD_APP_SVPT_ENFORCE 21
- IXSCAN 447

J

- join 18, 24, 60–61, 68

Joins 158, 463

L

look-aside capability 85
Lotus Extended Search 24
LOW2KEY 69, 72

M

MAX_QUERYDEGREE 48, 68, 74–75, 433
merge scan 467
merge scan join 83, 158
merge scans 60
missing or incorrect statistics/index information 170
missing or unavailable MQTs 210
model of different profiles of queries 427
MQT 24, 61, 82
MQT's 438
MQTs 25, 90, 436, 442, 452–453
MQTs/ASTs 85, 210
 functionality 86

N

nested loop 467
nested loop join 60, 83, 123, 158
network 112
nicknames 16, 70, 72
NLEAF 69, 72
NLEVELS 69, 72
NNSTAT 69–71, 73, 78, 111, 152, 155, 436, 467
non relational wrappers 16
non-pushdownable predicates 156
NPAGES 72
NUMERIC_STRING 68, 82–83, 95
NUMERIC_STRING nickname column option 47

O

on demand 2
 Automation 2
 definition 2
 Integration 2
 Virtualization 2
Online/event monitoring 166
Online/realtime event monitoring 36
Operation Merging 46
Operation moment 46
Optimized statement 451
Optimized Statement section 85

Original statement 451
outer join 111
Output streams 457
OVERFLOW 69, 72
overheads 164

P

Parallelism 160
partitioned database 26, 62
passthru 22
Passthru privileges 17
PDA 79, 95
performance considerations 24
performance factors 57, 59
performance management 32–34
 best practices 40
 hypotheses 39
 performance management cycle 34
 performance objectives 33
 proactive 32
 reactive 32
performance objectives 33–34
 Measurable 33
 Quantifiable 33
 Realistic 33
 Reasonable 33
poorly tuned sort heap and buffer pools 206
post threshold sorts 91
Predicate translation 46
predicates
 Index SARGable 94
private sorts 97
proactive approach 35
Problem determination methodology 37
problem resolution 40
PUSHDOWN 20, 68, 443, 467
Pushdown 156, 463, 467
pushdown 25, 45, 54, 60–61, 78–79, 110, 123
Pushdown Analysis 47
pushdown analysis 21, 25, 61, 78
pushdown factors 78
pushdown problems 272
PUSHDOWN server option 46
pushdownability 46, 467
pushdownable predicates 82, 156

Q

query fragment 52

Query Rewrite 430, 452
Query rewrite 46
query rewrite 24–25, 61, 112
 Operation Merging 46
 Operation moment 46
 Predicate translation 46

R

Redbooks Web site 633
 Contact us xx
referential constraints 89, 438
relational wrappers 16
remote data source related 161
remote query fragment 53
remote SQL fragments 45
replication 87, 437
request-reply-compensate protocol 45
RMTQTX 53, 102, 455
Routine monitoring 35, 165
RQRIOLBK 51, 53, 61, 68, 101
RTRIM function 81
runstats 73

S

sar 380, 390, 396, 415
SET SERVER OPTION 21
shared sorts 48
SHEAPTHRES 54, 68, 83, 91, 99
sheapthres 92
SHEAPTHRES_SHR 68, 83, 91, 99
SHIP 49, 53, 82, 102
SHIP operator 455
Snapshot Monitor 162
snapshot monitor 439
Snapshot Monitor switches 162
 Typical overheads 164
SORTHEAP 48, 54, 68, 83–84, 91–92, 96, 99, 113, 427
sorts
 non-overflowed sort 92
 non-piped sort 93
 overflowed sort 92
 piped sort 93
SQL Compiler 45
SQL compiler 451
 Check Semantics 45
 Parse Query 45
 Pushdown Analysis 46

Remote SQL Generation 50
 Rewrite Query 46
SQL dialect 50
SQL INSERT/UPDATE/DELETE 615
Statistics 43
 statistics 48, 61, 69–70, 72–73, 78, 109, 111, 123, 436, 467
 Statistics Update 69–71, 73, 78, 111, 152, 155, 436, 467
SYSCAT.COLOPTIONS 17
SYSCAT.COLUMNS 17
SYSCAT.FUNCMAPOPTIONS 17
SYSCAT.FUNCMAPPARMOPTIONS 17
SYSCAT.FUNCMAAPPINGS 17, 21
SYSCAT.FUNCTIONS 17
SYSCAT.INDEXES 17
SYSCAT.INDEXOPTIONS 17
SYSCAT.KEYCOLUSE 17
SYSCAT.PASSTHRUAUTH 17
SYSCAT.ROUTINES 17
SYSCAT.SERVEROPTIONS 16
SYSCAT.SERVERS 16
SYSCAT.TABLES 17
SYSCAT.TABOPTIONS 17
SYSCAT.TYPEMAPPINGS 17
SYSCAT.WRAPOPTIONS 16
SYSCAT.WRAPPERS 16
SYSSTAT 72
SYSSTAT.COLUMNS 18
SYSSTAT.INDEXES 72
SYSSTAT.ROUTINES 18
SYSSTAT.TABLES 18, 72
SYSTAT.INDEXES 18

T

timeron 454
topologies 26
traffic 53, 59
trusted mode 430
typical problem determination methodology 38

U

UNIQUERULE 71
User Mapping 23
user mapping 22

V

VARCHAR 79–80, 83, 95, 466, 469
VARCHAR_NO_TRAILING_BLANKS 68, 80–83, 469
VARCHAR_NO_TRAILING_BLANKS server option 47
VARCHAR2 80–81, 466, 469
views 452
Visual Explain 443
vmstat 380, 391, 396

W

Wrapper options 20
wrappers 13, 42, 51, 63, 430
 Communication with the data source 14
 data modelling 14
 Federated object registration 14
 Services and operations 14



DB2 II: Performance Monitoring, Tuning and Capacity Planning Guide



Redbooks

DB2 II: Performance Monitoring, Tuning and Capacity Planning Guide

DB2 Information Integrator V8.2 performance drivers and best practices

This IBM Redbook provides an overview of DB2 Information Integrator V8.2 key performance drivers; best practices to achieve optimal performance; and guidelines for monitoring a DB2 Information Integrator environment for capacity planning, problem diagnosis, and problem resolution.

Performance problem determination scenarios

This publication documents procedures for monitoring existing DB2 II implementations for the purposes of capacity planning. It also documents a methodology for routine and exception monitoring of a DB2 II environment for performance problem determination; and describes some commonly encountered performance problem scenarios and the step-by-step approach used in problem determination and resolution.

Capacity planning

**INTERNATIONAL
TECHNICAL
SUPPORT
ORGANIZATION**

**BUILDING TECHNICAL
INFORMATION BASED ON
PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks